2009-03

A validation metrics framework for safety-critical software-intensive systems

Cruickshank, Kristian John.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/4891

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A VALIDATION METRICS FRAMEWORK FOR SAFETY-CRITICAL SOFTWARE-INTENSIVE SYSTEMS

by

Kristian John Cruickshank

March 2009

Thesis Advisor:	James B. Michael
Thesis Co-Advisor:	Man-Tak Shing

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2009	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE A Validation Metrics Framework for Safety-Critical Software-Intensive Systems	**5. FUNDING NUMBERS**
6. AUTHOR(S) Kristian John Cruickshank	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Validation of safety-critical software requirements is a difficult and frequently misunderstood task. It answers the question of "*are we building the right product?*" and is essential to Software Engineering. However, validation is often confused with verification activities, or simply left as a final tick-in-the-box just prior to delivery. Current models for validation cannot satisfy the unique aspects of safety-critical software where "*building the right safety product*" is paramount. Software safety requires a new model for validation of safety requirements by proxy. The need for a proxy model becomes evident in the software safety process, where customer input for safety is reduced to the requirement of "*a safe system*."

This thesis defines a new proactive model for validation of safety-critical software requirements. Continuous assessment of validity of safety requirements is indicated by metrics as part of the Validation Metrics Framework. The generic framework combines the Goal/Question/Metric Approach with Goal Structuring Notation and then specializes in validation of safety-critical software. The metrics are measurements of safety products typical to safety-critical software development programs. A fictitious case study of a Rapid Action Surface to Air Missile is used to apply the framework, identifying the benefits of a proactive, indicative, validation technique utilizing a metrics framework.

14. SUBJECT TERMS Software Metrics, Safety Metrics, Validation Metrics, Metrics Framework, Validation, Safety-Critical Software, Software Engineering, Goal Question Metric, Goal Structuring Notation	**15. NUMBER OF PAGES** 144
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

A VALIDATION METRICS FRAMEWORK FOR SAFETY-CRITICAL SOFTWARE-INTENSIVE SYSTEMS

Kristian John Cruickshank
Flight Lieutenant, Royal Australian Air Force
BEng(Elec)(Hons), Central Queensland University, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2009

Author: Kristian John Cruickshank

Approved by: James B. Michael
Thesis Advisor

Man-Tak Shing
Thesis Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Validation of safety-critical software requirements is a difficult and frequently misunderstood task. It answers the question of “*are we building the right product?*” and is essential to Software Engineering. However, validation is often confused with verification activities, or simply left as a final tick-in-the-box just prior to delivery. Current models for validation cannot satisfy the unique aspects of safety-critical software where “*building the right safety product*” is paramount. Software safety requires a new model for validation of safety requirements by proxy. The need for a proxy model becomes evident in the software safety process, where customer input for safety is reduced to the requirement of “*a safe system*.”

This thesis defines a new proactive model for validation of safety-critical software requirements. Continuous assessment of validity of safety requirements is indicated by metrics as part of the Validation Metrics Framework. The generic framework combines the Goal/Question/Metric Approach with Goal Structuring Notation and then specializes in validation of safety-critical software. The metrics are measurements of safety products typical to safety-critical software development programs. A fictitious case study of a Rapid Action Surface to Air Missile is used to apply the framework, identifying the benefits of a proactive, indicative, validation technique utilizing a metrics framework.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF EQUATIONS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ACM	Association for Computing Machinery
BDUF	Big Design Up Front
BIT	Built In Test
C&C	Command and Control
CM	Configuration Management
EPSH	Estimated Percent Software Hazards
EPSSR	Estimated Percent Software Safety Requirements
ETA	Event Tree Analysis
FMECA	Failure Modes and Effects Criticality Analysis
FTA	Fault Tree Analysis
GQM	Goal/Question/Metric
GSN	Goal Structuring Notation
HAA	Hazard Analysis Achieved
HAS	Hazard Analysis Space
HAZOP	Hazard and Operability
HRI	Hazard Risk Index
IAP	Interface Adapter Panel
IFF	Identification Friend or Foe
IID	Iterative and Incremental Development
IV&V	Independent Verification and Validation
LCS	Launcher Control System
LIU	Launcher Interface Unit
MIA	Missile Interface Assembly
NASA	National Aeronautics and Space Administration
PHA	Preliminary Hazard Analysis
PHL	Preliminary Hazard List
PSH	Percent Software Hazards
PSSR	Percent Software Safety Requirements

RASAM	Rapid Action Surface-to-Air Missile
RMS	Root-Mean-Square
ROI	Return On Investment
SCU	System Control Unit
SDLP	Software Development Lifecycle Process
SHAD	Software Hazard Analysis Depth
SHCM	Software Hazard Criticality Matrix
SRM	System Reference Model
SSP	System Safety Program
SSRTM	System Safety Requirements Traceability Matrix
SSS	Software System Safety
SSSRTM	System Software Safety Requirements Traceability Matrix
UFC	Unique First Cause
UML	Unified Modeling Language
WCS	Weapon Control System

ACKNOWLEDGMENTS

I would like to express my gratitude to the Royal Australian Air Force for providing me with this excellent opportunity to extend my education and professional development. Without their support, this thesis would not exist.

I would like to thank my thesis advisors, Professor Bret Michael and Professor Man-Tak Shing. Your expertise, guidance, and motivation were integral to the research carried out.

Thank you to my friends and family, nearby and abroad. Your friendship and support is greatly appreciated. Thanks.

My deepest thanks go to my wife, Kate, for all of her support while I pursued full-time study far away from home, requiring her to put her career on hold. Thank you for everything you do for me.

Finally, thank you to my Lord and Savior Jesus Christ through whom all things are possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

> Not everything that can be counted counts, and not everything that counts can be counted.
>
> — Albert Einstein

A. OVERVIEW

As software engineers endeavor to design and build increasingly complex software systems, development and management techniques require commensurate levels of research to sufficiently cater for these increases. The role of software in safety-critical systems continues to grow in varying industries and applications; however, the tools used for necessary risk mitigation and management are lacking. Safe software is paramount in the defense industry, as large-scale weapon systems have the most potential for catastrophic unintended consequences. Ensuring that software functions do not contribute to system hazards in an unintended manner is largely an exercise of validation—identifying *correct* requirements to sufficiently mitigate hazardous situations. The problem addressed by this research is that of identifying invalid requirements for software safety. Metrics acting as indicators for validity of software safety requirements engender a proactive approach to investigation and identification of potentially invalid requirements.

Measuring the safety of software-intensive systems has only recently been investigated. There are currently no tools in existence for measuring the validity of software safety requirements. Metrics that indicate the validity of software safety are needed to cope with modern day safety-critical systems.

Validation of software safety requirements necessitates a new model of validation. Chapter III proposes a new model for validation of safety requirements, focussing on sufficiency of hazard identification, hazard analysis, and software safety requirements traceability, as a proxy for validation. This model forms the core of the proposed Validation Metrics Framework.

At present very little information exists on the use of metrics for the purpose of measuring safety. Even less, if any, information can be obtained on metrics for validation. Discussion on what validation metrics are, and how best to use them, is given throughout this thesis. By combining two popular software development tools (GQM and GSN) we have created a goal-based framework identifying a core set of metrics to aid in validating software safety requirements of safety-critical software-intensive systems.

The research and development of this thesis has resulted in a metric framework for validation of safety-critical software-intensive systems. There is currently no notion of validation metrics in open literature, much less a framework identifying purpose, application, and boundaries of the metric set.

Chapter II provides background information necessary to understand the context and overall direction of this thesis. It analyzes the current environment of software validation and identifies the gaps that exist when considering safety-critical software. This chapter defines the problem to be resolved.

Chapter III further explores the concept of validation of software safety requirements, identifying a new validation model. This chapter introduces a framework for validation metrics. It also introduces the core building-block of the framework—the Goal/Question/Metric (GQM) process for systematically identifying metrics.

Chapter IV details the development of the metric framework. The combination of GQM and Goal Structuring Notation (GSN) results in a hybrid goal-driven metric derivation specification language. Discussion of the purpose and boundaries of the framework is also given. The framework goal structure is developed along with questions and subsequent metrics. This chapter presents the proposed Validation Metrics Framework.

Chapter V investigates application of the Validation Metrics Framework through case study. The case study uses the Rapid Action Surface-to-Air Missile System (RASAM) as a representative fictitious safety-critical software-intensive system.

Chapter VI discusses the results of the Validation Metrics Framework and future work needing to be undertaken to further its utility and application.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. INTRODUCTION

As software complexity continues to increase in major defense systems, conveyance of stakeholder requirements, development to these requirements, and validation of these requirements has become exceedingly more difficult. This rate of growth in difficulty seems to be disproportionate to the techniques and methods that are used to ensure that the systems are developed to customer requirements and expectation. Safety-critical software-intensive systems require significant verification to ensure that they function as per requirements. Verification is only one portion of ensuring systems function correctly and is typically a well defined activity for software development. Validation is the other portion of ensuring that software is developed to the customer's satisfaction; however, is not so well defined. In the U.S. more software projects are cancelled due to incomplete requirements than any other factor.[1] Verification does not address this shortfall. Validation is the key tool to ensuring stakeholder requirements are sufficiently explored and met by the developed product.

Software metrics have typically been applied in the verification dimension because software validation was not well defined or understood. As software validation grows in maturity, so does its definition, tools, and techniques, including means for measuring the validation activity, its outputs, and impact on development. Software metrics are measurements of quality in product, project or process. They are used to make informed decisions about a particular aspect of a software project. Without measurement, there is no control. To effectively control software validation, it needs to be measured and presented through metrics.

[1] The Standish Group CHAOS Report, 1995.

This chapter provides the background information necessary for the following research. It identifies key definitions for safety-critical software-intensive systems, metrics, validation, and validation metrics, all necessary for understanding the context of this thesis. The current problem with validation will be described and the focus of this thesis will be defined.

B. SAFETY-CRITICAL SOFTWARE-INTENSIVE SYSTEM

1. Definition

MIL-STD-882D [1] defines safety as "Freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment."

Further, it also defines safety-critical as "A term applied to a condition, event, operation, process, or item of whose proper recognition, control, performance or tolerance is essential to safe system operation or use."

The Joint Services Software System Safety Handbook [2] defines safety-critical computer software components as "Those computer software components and units whose errors can result in a potential hazard or loss of predictability or control of a system."

In general terms, safety-critical software is software whose function either directly or indirectly influences a hazard or hazardous situation.

A software-intensive system is "a system where software represents a significant segment in system functionality, cost, development risk, or development time." [3]

Safety-critical software-intensive systems are those software-based systems that control, or provide input to, safety-critical (hazardous) applications.

2. Overview

Safety-critical software-intensive systems are becoming ubiquitous. Most modern aircraft are safety-critical software-intensive systems, as are many nuclear power stations, modern automotive systems, and even pre-tensioners on seatbelts. They rely on software for safe operation. Software systems are also continually growing in complexity at an increasing rate. Examples of this increasing complexity can be found in many modern day weapon systems that rely on a system-of-systems implementation to function. The web of relationships between individual systems comprising the complete system-of-systems becomes difficult to understand and analyze and will result in emergent behavior not considered in the stand-alone system. The safety-criticality of the software used in these systems can be overlooked, because software itself is not hazardous. Software is virtual—it cannot cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment. However, software is often directly responsible for the actions of hardware that can have unsafe effects.

Safety is a system property. Software cannot be analyzed for safety independent of the physical system it will control or influence. Even in the case of a pure software system, safety-critical decisions could be made based on the data it presents. Safety still remains a property of the system. Therefore, safety-critical software must be analyzed in the context of a software-intensive system.

C. VALIDATION

1. Definition

IEEE Std 1012-2004 [4] defines validation as:

> the process of providing evidence that the software and its associated products satisfy system requirements allocated to software at the end of each life-cycle activity, solve the right

problem (e.g., correctly model physical laws, implement business rules, use proper system assumptions), and satisfy intended use and user needs.

A more generic definition of validation is “the steps and the process needed to ensure that the system configuration, as designed, meets all requirements initially specified by the customer.” [5]

The latter definition allows for more flexibility in the interpretation of validation; it answers the question of “Are we providing what the customer wants?” However, both definitions assume the existence of correct requirements and focus on the concept of *building the right product* as defined by these requirements.

2. Overview

Software validation has always been problematic in software engineering. Unlike many other engineered products, software often cannot be visualized, thus, in many cases, resulting in software validation being a reactive last minute process. It is hard for validation to be a continuous process throughout development as with many physical systems when there is no physical product to monitor. The ultimate validation of software systems is effectively the operational evaluation of the system by the user, and often this is where validation of the system is relegated. The IEEE definition lends itself to this approach as it focuses on the satisfaction of requirements through testing. Quite simply, validation of software has not had the same rigorous research and development of processes as other areas of software engineering, particularly verification. A simple search[2] of the IEEE Xplore database for articles with *software* and *validation* in the abstract reveals sixty articles. The same search for *software* and *verification* shows 110 articles. Validation is a relatively misunderstood process. As a result of this, of the sixty articles found, roughly 30% address the essence of validation

[2] IEEE Xplore Database, accessed 07 Jun 2008.

as discussed below.[3] Most of the articles mistakenly combine verification and validation as one element, term verification as validation, or confuse assurance and quality activities as validation. Although this is a very simplified example, it does help to portray the seriousness of what is a critical aspect of software engineering.

The software engineering discipline has become competent in the area of verification. We can build portions of systems to their applicable specifications with relative success. However, we still build systems that do not meet customer's expectations and requirements. One of the key tools to address this situation is validation. Significant efforts afforded to software validation are now a priority for software engineering. More proactive, rather than the typical reactive, solutions are being sought. The IEEE definition of validation indicates that it is a process to be carried out at the end of each phase (or lifecycle activity); however, this should only be the finalization, or completion, of validation. The validation process should be a proactive and continuous process to be carried out prior to, and in parallel with, the development and verification activities with closure at the end of each phase.

Although validation focuses on ensuring that initial customer requirements are met, there is more to validation than meets the eye. Validation is required whenever a requirements derivation process occurs (i.e., a translation of requirements from one domain to another). An example of this is taking a customer's requirements in their natural language and translating them into a specification. The specification needs to be validated to ensure that it maps back to the cognitive understanding of the stakeholders who originally supplied the requirements [6]. To ensure the traceability of products for validation, the validation process is ongoing throughout the development cycle whenever this translation of requirements takes place. Any higher-level requirement being

[3] Thirty percent is an optimistic estimate. Roughly 10 of the 60 articles did not concern validation of software; rather they addressed validation of hardware systems through the use of software models.

translated to a lower-level requirement requires a validation process to ensure that the products of the lower-level requirements are indeed valid. In contrast, verification is defined as “The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [7]

The key difference between validation and verification is that verification simply ensures that requirements for a given phase are met. Validation ensures that overall customer requirements (i.e., customer *expectations*) are met. There is somewhat of an overlap in validation and verification processes, particularly when considering either process in the “middle-levels” of abstraction. They are in fact processes orthogonal to one another. In its purest form, validation ensures that customer expectations are met; failure to meet these expectations (assuming they are constant throughout the project) indicates a failure in the validation process. Figure 1 shows graphically how verification and validation contrast.

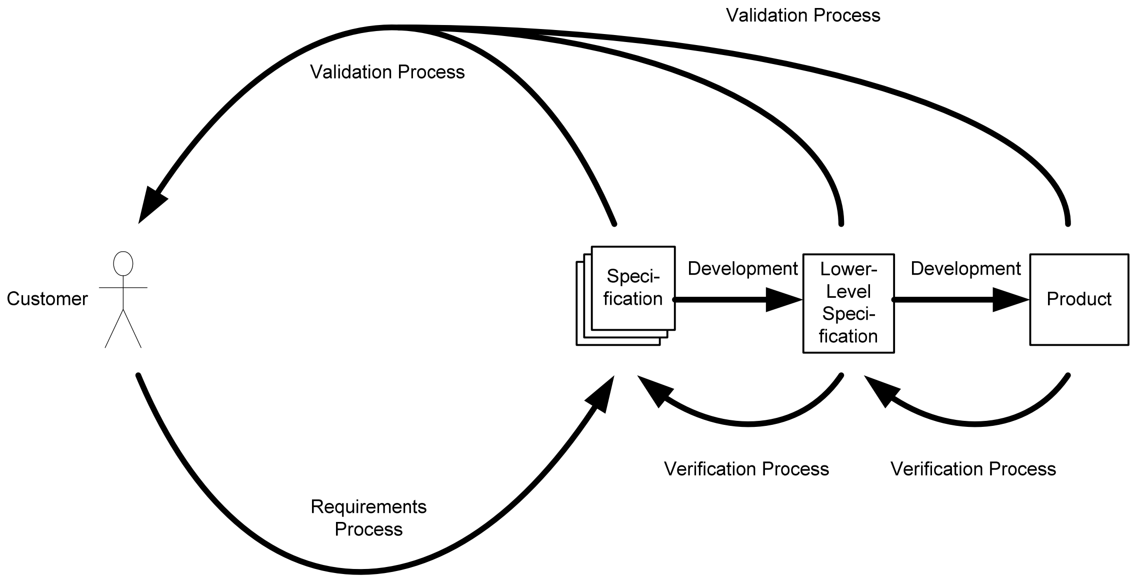


Figure 1. The Verification and Validation Process

The end-result of any validation process should be actionable data presented as feedback to the many different customers, or stakeholders, of the system. This effectively creates a feedback loop from any stage in the development process that allows the customer to clarify their expectation of system behavior.

D. METRICS

1. Definition

IEEE Std 610.12-1990 [7] defines a metric as “a quantitative measure of the degree to which a system, component or process possesses a given attribute.”

Further, a software quality metric is “A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute. [7]

Software metrics are therefore quantitative measurements of either product (system or component), process, or even project (in this case software projects) indicating the quality of a desired attribute. However, software metrics can be concerned with more than just quantitative measurements. Since we are measuring quality of product, process, or project, qualitative aspects must be considered. Metrics can also be qualitative in nature.

2. Overview

For the purpose of this thesis, metrics are measurements of quality. In most well established engineering disciplines candidate measurement attributes are well understood. Software engineering is one of the disciplines outside of the “most” category. Software engineering is a young engineering discipline and as such does not have the hundreds of years of empirical scientific foundation as other core engineering disciplines. Partly due to this fact, many measures of software are not well understood or are ill-defined.

Measurement:

> in the most general sense, is the mapping of numbers to attributes of objects in accordance with some prescribed rule… The mapping must preserve intuitive and empirical observations about the attributes and entities [8].

Empirical observation requires experimentation. Therefore, strictly speaking, for a software metric to be valid it must be based on empirical observation through experimentation, whether qualitative or quantitative.

Often with software development, the single best metric is sought. Given the above definitions, such an approach comes across as foolish. An engineered product cannot be properly understood purely through the application of one measurement. A single software metric can only present a single view of a software product. Multiple metrics are required to sufficiently understand a product or process. Knowing a car's weight gives no indication of overall performance, it is only one piece of the picture. It is also possible that the gathered metric does not answer a question of interest or provide any decision-support value. The same is true for software.

Many definitions of metrics require them to be quantitative. However, many qualitative measures of product, process, or project are also valid. Consider a fast car. The "fast" attribute indicates its relative quality. It is a qualitative metric, understood in a qualitative manner. Qualitative metrics are relative to some quantitative measure; however, exact position on the quantitative scale may not be possible with the given data due to the use of a weak ordering. A qualitative metric still provides relevant indication of quality. This is an important aspect of metrics for software since the abstract nature of software can often preclude quantitative measures of certain quality attributes.

E. VALIDATION METRICS

1. Overview

Many of the metrics currently used in software engineering focus on verification aspects of the software process or product. These metrics are concerned with measurements that ensure requirements for particular phases are met. Metrics such as fault density, test coverage, etc., are typically used for verification purposes as they focus on given requirements. Just as there are measurements for all different aspects of a physical machine—metal density, fatigue rating, size, safe working load, etc.—software products need to be described through different types of metrics. Validation metrics are measurements of product from the aspect of fulfilling customer requirements and expectations.

Validation metrics must focus on measurements that can be used for the validation process. Since validation is concerned with ensuring that the customers requirements and expectations are met (*building the right product*), validation metrics should tie back to the most abstract requirements in some way. In essence, validation metrics should provide measures of software quality, indicating the fulfillment of customer requirements, to the customer as actionable data. Verification metrics and validation metrics should be complimentary, i.e., they should both contribute to the overall understanding of the system, but they will focus on distinctly different aspects of the system.

According to Munson [8], measurements can be divided into two categories for software engineering: primitive and derived. Primitive metrics are presupposed by no other, they are direct measurements of product attributes. There are no assumptions made by primitive metrics to arrive at their result. On the other hand, derived metrics are those that are not direct measurements. They are derived from assumptions or a combination of metrics. McCabe’s Cyclomatic Complexity is an example of a derived metric. Although the underlying attributes of Cyclomatic Complexity (number of edges, nodes and connected components) are primitive metrics (although in a virtual sense), the end result (Cyclomatic Complexity) is a derived metric. Caution must be taken when dealing with derived metrics. In a sense, the adage of “adding apples to oranges” is of concern. Because many primitive metrics of software will be based on something that is not physical, it can be quite tempting to combine primitive metrics that may in fact have no relationship. As validation metrics will typically be more abstract than verification metrics, in the sense that the validation of the metric itself relies on abstract software artifacts, this issue must be sufficiently managed. Derived validation metrics require significant forethought and analysis to ensure that they do not portray invalid measurements.

F. RELATED WORK

As discussed above, software validation has not been afforded the same levels of research and development as many other aspects of software engineering. As such, there is little literature addressing the concept of validation metrics. In fact, an extensive search of the ACM and IEEE Xplore databases revealed no published literature on metrics for the purpose of validation. The following paragraphs, however, detail some of the more recent advances on metrics and validation that aid in the formation of a validation metrics framework.

1. Metrics

Whalen, Rajan, Heimdahl, and Miller [9] present a new coverage metric for requirements-based testing. They define the Unique First Cause (UFC) metric that determines coverage of high-level formal requirements. The method utilized allows for automatic generation of test cases based on high-level requirements, and a by-product of these test cases is the coverage metric. Whalen et al. claim that the UFC metric provides “objective, implementation-independent measures of how well a black box test suite exercises a set of requirements.”

The results of the case study are encouraging, and the independence of implementation would aid in the overall independent verification aspect of many safety-critical systems. However, Whalen et al. claim that the metric “integrates and crosschecks several of the verification and validation activities.”

Although the metric is certainly useful for verification, there is no indication that it provides actionable data back to the customers and stakeholders based on their requirements. It does link back to the highest-level *formal* requirements, remaining independent of any implementation-specific requirements, but does not make the necessary link back to customer requirements for it to be considered validation at the highest level. The translation from user documentation detailing their requirements and expectations to the formal requirements indicates that there is still a gap to be bridged back to the highest-level requirements.

Tasiran and Keutzer [10] detail a number of different coverage metrics for the use of functional validation of hardware design. They provide an extensive list of coverage metrics that can be used for software simulation models of hardware designs. By utilizing software simulations Tasiran and Keutzer claim that “Coverage metrics ensure the optimal use of simulation resources, measure the completeness of validation, and direct simulations toward unexplored areas of the design.”

The context of this “validation” needs to be explained to fully understand why this form of validation may be applicable to hardware design in this domain, but generally not for software-intensive systems. There is a distinct difference between most customers of commercially developed hardware and safety-critical software-intensive systems. Essentially, for the hardware systems that Tasiran and Keutzer are dealing with, the customers are the developers. There may be a level of independence within their organization, however the organization as a collective ultimately decides upon the behavior of the hardware and how it will be designed. The domain barrier between customer and developer is somewhat blurred and it is likely that the customer, in this context, is able to “speak” in the language of the developer. There is no major language translation; therefore, formal software models of the hardware products do provide actionable data back to the customer when utilizing common software verification metrics. The same is not true for most safety-critical software-intensive systems. Any models of the software-intensive systems that are to be used for validation must be sufficiently abstract such that the customers and stakeholders are able to understand and act upon data directly from that model.

The metrics that Tasiran and Keutzer present are in fact the typical verification metrics used throughout the software development process. As with the Whalen et al. paper, these metrics do not provide customers and stakeholders of safety-critical software-intensive systems with actionable data based on their requirements.

2. Validation Models

Pingree et al. [11] expand on current software modeling techniques for the validation of mission-critical software design and implementation by focusing on software models consisting of statecharts. The process provides an independent understanding of the system being designed by creating a statechart specification from which all “validation” steps are taken. In effect, the statechart specification acts as a reference model for the remaining development. Through this method, Pingree et al. claim, “we are now able to specify and validate portions of mission critical software design and implementation using exhaustive exploration techniques.”

However, there is no mention of feedback to the customers and stakeholders. Statecharts provide a necessary higher layer of abstraction for complex systems, but the focus on this technique is model correctness, not customer feedback, (i.e., validation). It serves as a model for validation in the following phases of development, and does aid in presenting a model that customers and stakeholders are more easily able to comprehend; however, the customer feedback portion is still not clearly identified. Rather than using traditional testing techniques to verify the design once the product is complete, a pro-active approach is being taken to ensure that the developer’s understanding of the design is correctly implemented (essentially building the right product from the designer's point of view through to the implementation) and that no critical design errors are made. Design errors in this sense focus on correctness through exhaustive model exploration, not necessarily building "*the right product*." This technique still has a high level of focus on building "*the product right*," i.e., verification.

Drusinsky et al. [6] present a framework for computer-aided validation for use by Independent Validation and Verification (IV&V) teams. This technique focuses on an executable System Reference Model (SRM) using formal assertions to specify mission and safety-critical behaviors. The SRM is independent of the actual system under development and the focus of the paper is on the IV&V team as the customer, or stakeholder, for validation purposes. It is a pro-active approach to ensuring the IV&V team has a firm understanding of the desired system behavior both prior to and during development of the system. Drusinsky et al. state:

> The IV&V team's independent requirements effort should develop a description of the necessary attributes, characteristics, and qualities of any system developed to solve the problem and satisfy the intended use and user needs. The IV&V team must ensure that their cognitive understanding of the problem and the requirements for any system solving the problem are correct before performing IV&V on developer-produced systems.

Hence, the focus of this technique is on customer and stakeholder requirements. The SRM consists of use cases, Unified Modeling Language (UML) artifacts and formal assertions derived from the user requirements to describe the behaviors of the system. By focusing on the UML artifacts and high-level use cases, the IV&V team is ensuring that customer requirements are central to developing their understanding of the system through statechart assertions. The assertions are created from reified high-level user requirements and implemented in code to assess predictions of how the system should and should not behave. Thus, there is a one-to-one mapping between each reified user requirement and its corresponding assertion. This approach allows not only the IV&V team to validate the model of requirements, but also the requirements posed by the stakeholders.

This technique assists in validation primarily from the perspective of the IV&V team. Validation from the different customer and stakeholder perspectives can vary depending on their level of involvement in development and understanding of different techniques used. However, the framework does form a firm grounding for subsequent validation metrics for IV&V and will also contribute for validation metrics for other customer and stakeholder domains. Providing metric data back to the customers and stakeholders through use of the SRM approach will further ensure that the validation loop is correctly carried out.

G. THE PROBLEM

Safety-critical software-intensive systems are becoming increasingly complex as technical boundaries are overcome and each prior level of complexity (in terms of functional dependencies, relationships and difficulty of requirements validation) is better understood. However, with each increase in engineering ability comes further complexity and potential ambiguity between customers/stakeholders and developers, which is currently not being sufficiently dealt with. Validation is the key tool for ensuring that systems are developed in accordance with the requirements of customers and stakeholders. Validation is only one tool for battling the complexity issue by ensuring that "*the right product is built*." However, how do we measure the effectiveness of validation, or provide metrics to aid in the validation process? Very little, if any, research exists for the application of metrics for validation purposes. Without metrics for validation, estimates of success are very subjective. Software engineering requires that as little subjectivity as possible be introduced when dealing with the validation of safety-critical software-intensive systems—validation metrics are one tool for reducing that subjectivity. Metrics as feedback to customers and stakeholders are essential in ensuring that the system is correctly understood, particularly in hazardous applications. Combined with proactive validation techniques (rather than using operational evaluation as the “tick-in-the-box”), such as the SRM approach, validation metrics will aid in the validation process to increase the likelihood of successful development of the right software-intensive systems.

Software safety in itself is highly subjective in nature when considering validation. Often the safety of a system cannot be tied back to customer expectations or requirements, other than the fact that they wish to have a *safe* system. Traceability and validity of software safety requirements requires a different approach to validation than the traditional matching of system specification to customer expectations and requirements.

The following chapters will detail measurable characteristics of the validation process for safety-critical software-intensive systems and present guidelines for a validation metrics framework. The goal of this framework will be to provide actionable data to customers and stakeholders for the purpose of validating safety artifacts.

THIS PAGE INTENTIONALLY LEFT BLANK

III. VALIDATION METRICS FRAMEWORK ANALYSIS

A. INTRODUCTION

The previous chapter introduced the concept of validation metrics and provided the background information necessary to understand the scope of this thesis. This chapter will further analyze the validation metric concept to determine required characteristics and define the objectives of a framework necessary to identify applicable existing metrics or the characteristics of new metrics. Artifacts suitable for feedback to the necessary stakeholders will also be examined as part of the framework. The chapter will focus on safety-critical software-intensive systems. However, development of a framework for validation metrics is not limited to this scope.

B. VALIDATION AUDIENCE

To successfully analyze and define the characteristics that a validation metrics framework must possess, a more in-depth look at validation is required. The previous chapter detailed the broad concept of validation; however, there is again more to be realized, particularly when considering the intended validation audience.

1. Relevance of Data

As mentioned previously, validation is mandatory whenever a requirements translation process takes place. A validation process is successful when a reverse-translation of requirements occurs, and the originator of the requirements can validate the subsequent artifact(s). This typically takes place by the originator testing the resulting product of the translation and observing system behavior. An important key aspect of validation is that it is not restricted to the highest level of stakeholders. Indeed a validation process can, and should, be applied for every level of translation. However, as the requirements and design are translated further and further from the original stakeholders' domain, the more perplexing and foreign they become. This is one of the key difficulties in using formal methods[4] as the sole vehicle for conducting validation. The intended users and high-level stakeholders cannot understand the language in which the system is being portrayed. Instead, relevant data needs to be extracted and presented to the users and high-level stakeholders in simple terms and in the language of their domain. Therefore, for any validation technique, *relevance of data* must be determined. The output of a validation metrics framework must allow for data to be presented to the intended audience in a meaningful and relevant manner.

2. Users and Stakeholders

In the previous chapter, validation was presented in terms of customers and stakeholders. In reality, customers and stakeholders can be more clearly defined as users and stakeholders. Users concerned with the validation of a safety-critical software-intensive system are primarily the recipient of the system, with pre-determined expectations based on their conveyed requirements. They are usually easily identifiable and are the ultimate "validator" of the system. However, the realm of stakeholders can be difficult to bound and can be involved at many levels of system development. In this thesis, stakeholders can be defined as "*other than user.*" Essentially every person involved in the development of the system is a stakeholder. Therefore, stakeholder requirements and expectations will also have a validation aspect to them, which will not necessarily be the same as that of the user. Based on this, an addition to Figure 1 is required to more accurately reflect validation. This line of thought is equivalent to that used in Barry Boehm's WinWin Spiral Process Model [13], whereby different stakeholders are identified for different levels of development. Particularly in the case of contractor developed software-intensive systems, a

[4] Mathematically based techniques for describing system properties. For more information see [12].

validation process is required from one development domain to another. Users will often be involved at the lower levels of development, with corresponding relevant data, but to ensure traceability of high-level requirements to design and validation of these requirements, validation steps must be carried out against each intermediate level of requirements. Figure 2 shows the additional validation pathways that can occur throughout the development of a software-intensive system. In this figure, it is shown that users/stakeholders are not always concerned with the “middle” levels of development; instead, this is left to the domain of developers (whom are also stakeholders as per the WinWin model).

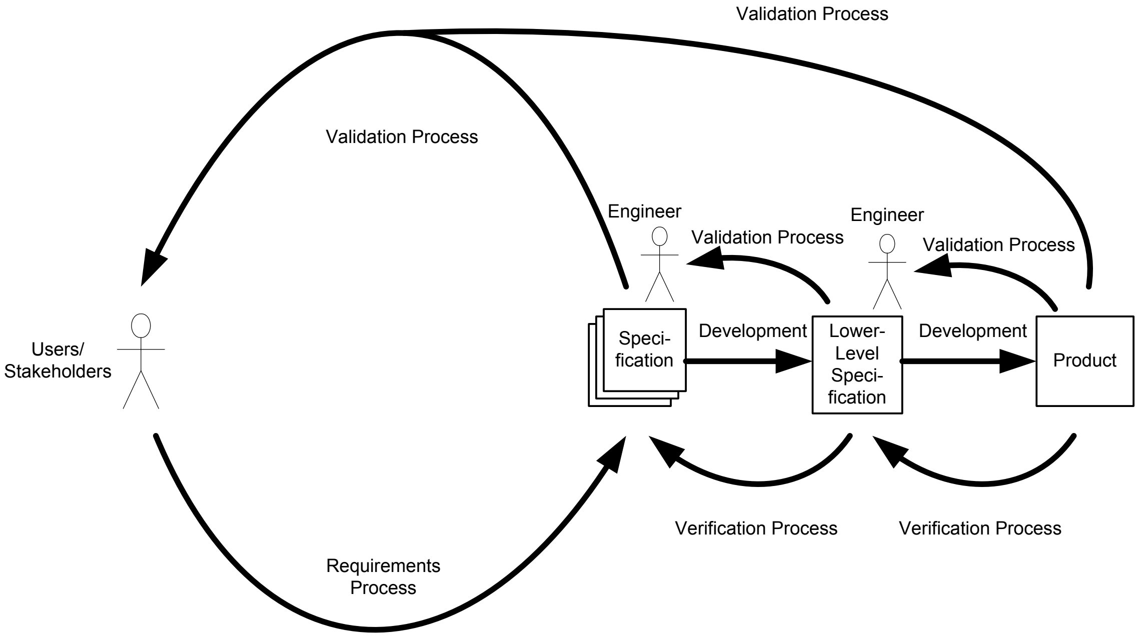


Figure 2. Modified Verification and Validation Process

From the above diagram, it is clear that validation metrics must not only be applicable to the users of the system, they must also be relevant to the stakeholders of the system. One metric cannot satisfy all the requirements of users, auditors, project managers, developers, regulatory bodies, etc. Different aspects of product, process and project will be relevant to different people. As per the previous section, relevance of data is a key attribute for developing validation metrics. Now armed with a more thorough understanding of validation, where it exists, how it can be used, and who uses it, a framework for identifying and developing suitable metrics can be more readily defined.

3. Intended Audience

The validation audience does not necessarily include all stakeholders of a system. Instead, the validation audience is comprised of those stakeholders concerned with ensuring that requirements are correct, according to user expectations. This statement in itself is subjective, and often the definition of *stakeholder* can become unbounded as everybody attempts to obtain a stake in the system design. However, to bound the scope to a somewhat realistic manner, Figure 3 shows a use case diagram for the possible users of a validation metrics framework. The Mission Assurance team displayed below is a broader group than simply the V&V team as it includes all stakeholders concerned with assurance of the system (V&V, IV&V, Auditors, etc.). Operators and owners are those referred to as users or customers in previous sections.

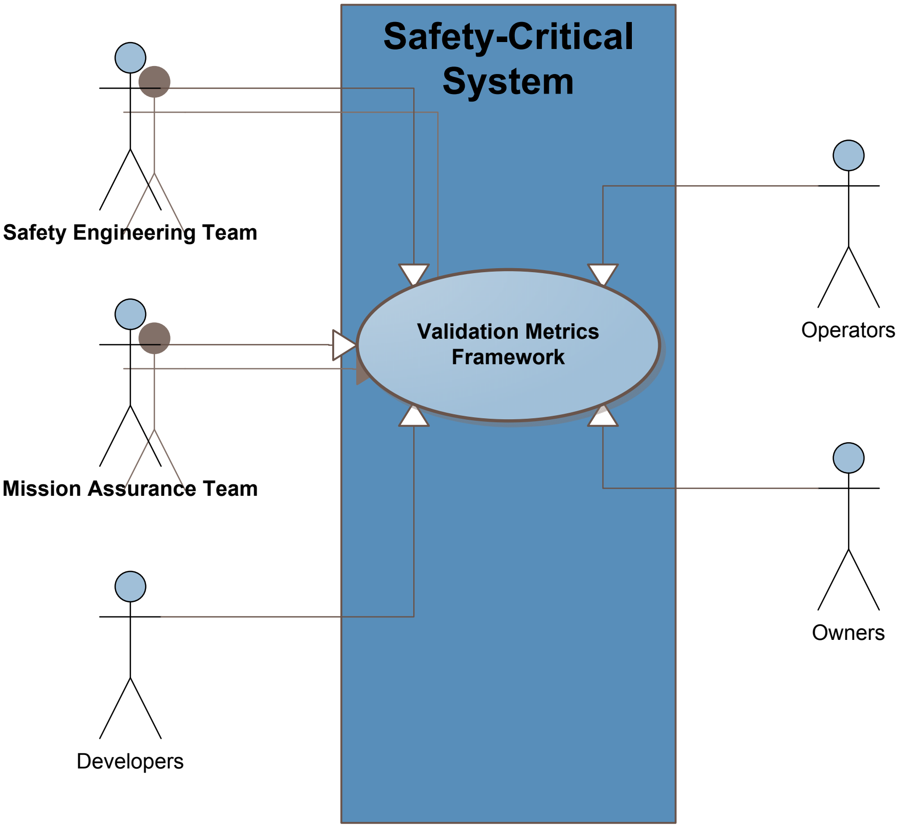


Figure 3. Validation Metrics Framework Users

The target audience of this Validation Metrics Framework will be the Safety Engineering team (shown in bold). Although the framework will be applicable to the other users and stakeholders shown, it will be tailored in this instance to the safety engineering team's domain. Actionable data arising from the framework will be primarily specific to the safety engineering team's scope, based on user and stakeholder input in the form of use cases, high-level requirements documents, and potentially user feedback on the artifacts produced from the framework. However, ultimately validation ends with the operators and owners of the system; therefore the safety engineering team will effectively act as a proxy for the framework—the metric data must also reveal safety aspects of the system to the owners and operators. Developing a framework to a much broader application of validation could result in inadequate explanation for application to real world scenarios, and therefore will be pursued as future work after successful definition and application in the safety engineering domain. The Mission Assurance team (also shown in bold) will effectively be the appliers of the framework. They will be responsible for obtaining the metrics data on behalf of the safety engineering team.

C. THE REALITY OF VALIDATION

Up to this point, validation has been described in idealistic terms. In reality, validation cannot exist as an isolated entity. To validate a system requires more than just carrying out a "validation" procedure. Validation is essentially the culmination of many other development processes. Although it is possible to perform these other processes (verification, requirements analyses, etc.) without addressing validation, the reverse is not true. Essentially all of these processes, including validation, comprise software assurance. By showing that best practices for the development of software have been used, and safety of the system can be sufficiently argued, the confidence that can be placed in validation is increased. A higher level of software assurance assumes a higher integration of validation; however, it is often not addressed to the same level as other development procedures. Verification is simply required for any system, but in terms of validation, verification is an underlying assurance. If requirements can be verified through testing at each level of development, there can be more confidence placed in the validation process carried out, as it can be assumed that stakeholder requirements and expectations are more closely matched to the actual developed product.

Therefore, in the following paragraphs and chapters, validation will not be treated as a completely separate entity. Some metrics identified through the framework may be considered verification or project metrics (there may not be any characteristic that could be identified as "validation"), but they form a crucial part in understanding the system from the validation perspective. In this manner, it will be possible to leverage metrics that may be collected even without applying a validation metrics framework. However, by applying this framework the existing metrics are given more meaning when trying to conduct validation.

D. SOFTWARE SAFETY

Safe software, contrary to various opinions and understandings, cannot be achieved through software reliability practices [14]. Software reliability is defined as “the ability of a system or component to perform its required functions under stated conditions for a specified period of time.” [7]

The focus on reliability is performance according to requirements in terms of failure to meet those requirements. Although reliability can be used as an indirect indicator of safety, lack of hazard analysis and subsequent safety requirements can render this a moot point, as reliability does not mandate *safety requirements*. As discussed by Leveson [14], most safety-critical failures can be traced back to incorrect requirements, i.e., a lack of understanding as to what the software should do under hazardous conditions. In essence, this falls under the validation domain based on stakeholder expectation. Stakeholders expect the system to be safe without necessarily providing specifications for safety. As further evidence to the validation case Leveson [14] states that, “although coding errors often get the most attention, they have more of an effect on reliability and other qualities than on safety.” This statement indicates a reliance on verification according to requirements, rather than validation according to expectation, as being a key *quality* improvement technique. Validation, again, is not afforded the attention it requires.

Therefore, safety-critical software-intensive systems require a systematic metric framework to aid in validation. Although much effort has been afforded to developing hazard analysis techniques and hazard reduction techniques (such as Failure Modes and Effects Criticality Analysis, Event Tree Analysis, Fault Tree Analysis, etc.), there is little to no evidence that measurements of these processes and products are being conducted to aid in answering the question of “*are we building the right safety product?*” (i.e., validation of system safety).

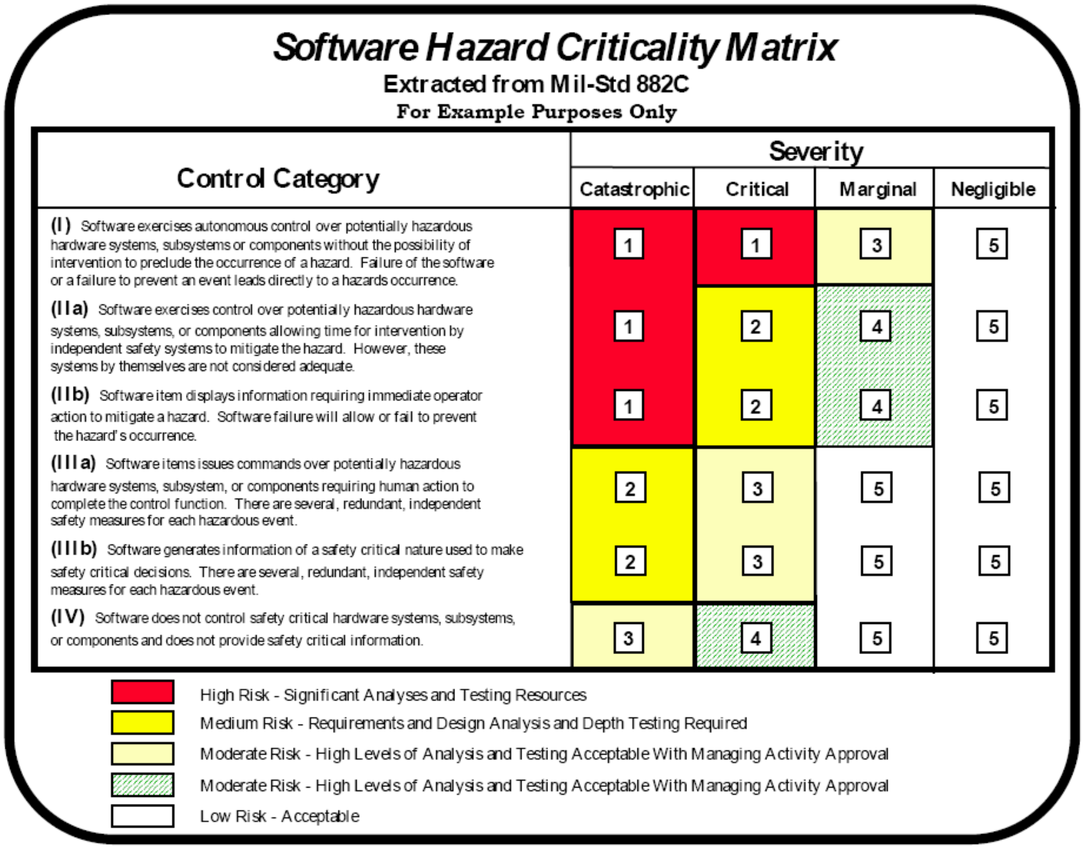
Safety requirements can be divided into two categories: generic requirements and system specific (or derived) requirements. Generic requirements are those recommended in standards, contained in workplace procedures, or identified in “lessons learned,” etc. They are essentially good practice, based on previously identified common causes leading to known hazards, to aid in developing a safer system. Derived requirements are those that are realized through the undertaking of hazard analysis and association of software functions that may contribute to identified hazards. These derived safety requirements may be more specific to the validation of safety-critical software-intensive systems than any other artifact or product, as high-level user documentation often does not provide such detail.

There are many products and procedures involved in the engineering of a safety-critical software-intensive system. Hazard identification, hazard analysis, safety-critical software function identification, and verification of safety requirements are some of the areas that will need to be considered in the development of a validation metrics framework. These products and processes will be some of the major foci of the Validation Metrics Framework for safety-critical software-intensive systems.

1. Software Hazard Risk Assessment

Unlike risk assessment for hardware, software risk assessment has unique qualities that inhibit the traditional assignment of consequence/severity and likelihood/probability. Determining the probabilistic nature of software is a hotly debated topic in the software engineering discipline; however, for the purpose of this thesis the assumption is made that software failures are systematic. That is to say, they are caused by incorrect requirements (design errors) or development errors, therefore are systematic in nature and cannot be assigned probabilistic failure rates. Although this is contrary to much of the field of software reliability, it does allow for the use of many pre-conceived software safety tools.

Determining the safety risk associated with software requires a different approach to hardware. A typical approach[5] to determining software risk is to determine the software's level of control over the associated hazard or hazard causal factor rather than determining the probability (or likelihood) of a hazard/hazard causal factor occurring. Figure 4 shows a Software Hazard Criticality Matrix (SHCM) for assessing the risk of software contributing to system hazards.



Software Hazard Criticality Matrix
Extracted from Mil-Std 882C
For Example Purposes Only

Control Category	Severity			
	Catastrophic	Critical	Marginal	Negligible
(I) Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazards occurrence.	1	1	3	5
(IIa) Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.	1	2	4	5
(IIb) Software item displays information requiring immediate operator action to mitigate a hazard. Software failure will allow or fail to prevent the hazard's occurrence.	1	2	4	5
(IIIa) Software items issues commands over potentially hazardous hardware systems, subsystem, or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.	2	3	5	5
(IIIb) Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.	2	3	5	5
(IV) Software does not control safety critical hardware systems, subsystems, or components and does not provide safety critical information.	3	4	5	5

🟥 High Risk - Significant Analyses and Testing Resources
🟨 Medium Risk - Requirements and Design Analysis and Depth Testing Required
▭ Moderate Risk - High Levels of Analysis and Testing Acceptable With Managing Activity Approval
▨ Moderate Risk - High Levels of Analysis and Testing Acceptable With Managing Activity Approval
□ Low Risk - Acceptable

Figure 4. Software Hazard Criticality Matrix [From [2]]

Figure 4 utilizes the Control Category scheme given by MIL-STD-882C. With regards to the Control Category schemes, The Joint Software System Safety Handbook [2] states that, "The SSS [Software System Safety] team must

[5] As given in MIL-STD-882C and RTCA DO-178B.

review these lists and tailor them to meet the objectives of the SSP [System Safety Program] and software development program." For the purpose of this thesis, the Control Category scheme of MIL-STD-882C, as presented in Figure 4, will be assumed.

The Joint Software System Safety Handbook [2] emphasizes the fact that the SHCM is not intended to be used directly as a Hazard Risk Index (HRI) matrix. Because it is not possible to assign a probability of occurrence, the risk assessment provided by the SHCM is not entirely compatible with the risk assessment of a HRI. Instead, the SHCM reflects risk in the unique terms of software, indicating a level of rigor required to address the risk level. In some cases, the risk level may warrant an alternative solution that does not utilize software control. Therefore, when determining hazard risk that includes software causal factors, engineering judgment must be applied to determine a level of probability, taking into consideration the SHCM rating, the level of rigor applied, and the resultant safety measures developed.

2. Hazard Causal Factors

Software hazards are in and of themselves causal factors to system hazards. As discussed earlier, software cannot create a mishap by itself. However, software is often responsible for system functionality that can create mishaps. The linkage of system hazards to software hazards (causal factors) requires an in-depth understanding of system functionality. Software functionality contributing to system hazards is identified as a first-order causal factor. They are in themselves hazards, but with further analysis second and even third-order causal factors can be revealed. Analysis of software causal-factors beyond first-order is typically only required for medium- and high-level risks as identified in the SHCM. Although this is a generally accepted rule of thumb, and in this thesis assumed to be the norm, it may be the case that certain industries, or applications of software have determined a different level of analysis. Therefore, any metric framework measuring the depth of software causal factor analysis must be tailorable to the different standards of measurement of sufficiency.

For the purpose of this thesis software hazard causal factors, since they are in fact hazards themselves, will be referred to as software hazards. Any reference to software hazards must be taken in context, but generically they will always be causal factors. It is also assumed that medium- and high-level software hazards require analysis to the level of third-order causal factors, unless justified otherwise.

E. VALIDATION OF SOFTWARE SAFETY REQUIREMENTS

Validation of safety requirements is not achieved through metrics alone. Metrics, applied through this framework, will *aid* to validate software safety (and hence, system safety). However, an understanding of validation in the software safety context is required.

Validation is largely dependent on the audience. Since we have defined the subject audience, determining the scope of validation is now possible. The primary audience is the safety engineering team; however, their focus on validation is two-fold. They wish to ensure that safety requirements are valid from their perspective, but also that validation is performed in accordance with high-level stakeholder requirements and *expectations*. Expectation is derived from stakeholder requirements (through documents such as statement of needs, use cases, concept of operations, operation requirements documents, etc.) as a foundational starting point; however, stakeholder feedback clarifies our understanding of this foundation. Therefore, validation of safety requirements needs to provide actionable data to not only the safety engineering team, but also to stakeholders. The following paragraphs will break down the different elements of validation that this framework will address.

From the safety engineering team's point of view, requirements validation is key—that is, software safety requirements validation. According to Weaver

[15], “Requirements Validation—Demonstration that the set of Safety Requirements is complete [sufficient] and accurate.”

Although safety requirements completeness is desirable, determining the completeness of the set of safety requirements is not realistic as pointed out by Gödel [16]. Therefore, where Weaver describes “complete” requirements, it will be considered as “sufficiency” of requirements. He further explains the concept as, “Requirements validity is demonstrated through the thoroughness of the approaches used for hazard identification.” [15]

Using the above definitions, it can be derived that validation of software safety (for the safety engineering team) is primarily concerned with software safety requirements validation. Although Weaver’s statements above indicate that requirements validation simply requires that hazards identified are sufficient and accurate, software safety validation requires that: system hazards identified are sufficient, the software interaction with these hazards is known and accurate, and the subsequent derived software safety requirements are also sufficient and accurate. In reference to determining how sufficient and accurate these safety requirements are, Weaver [15] states, “It [requirements validation] necessitates demonstration that a suitable level of effort, expertise and knowledge has been applied to the failure mode identification.”

Essentially, the above definition means that evidence for the validity of software safety requirements can be shown through the thoroughness of the safety process. For software safety requirements to be valid, hazard identification must be measured and hazard analysis must be measured.

In his thesis, Weaver separates requirements validation from requirements traceability. However, traceability of software safety requirements is an important component of validation. Traceability is a pre-requisite of a valid requirement. Therefore, traceability from software safety requirements must also be measured. Traceability of software safety requirements is not as simple as traditional traceability of system requirements. Traceability of all safety requirements subsequent to the initial derived safety requirements must be ensured, but traceability from derived safety requirements to system requirements must also be complete. In this case completeness means that there are no “orphan” safety requirements, i.e., there are no safety requirements that cannot be traced back to a hazard or, through a level of indirection, to another safety requirement. This is accomplished by tracing upward from derived safety requirements through the hazard analysis process to identified hazards, which are based on system requirements. Essentially, every software safety requirement must be traceable to an identified system level hazard (often through intermediary hazard causal factors). This process ensures traceability to the highest-level stakeholder requirements. One way this could be achieved is through dependency graphs in Configuration Management (CM) tools—automating this process would provide a simple and efficient solution to traceability of safety requirements. Figure 5 shows how traceability of safety requirements is achieved.

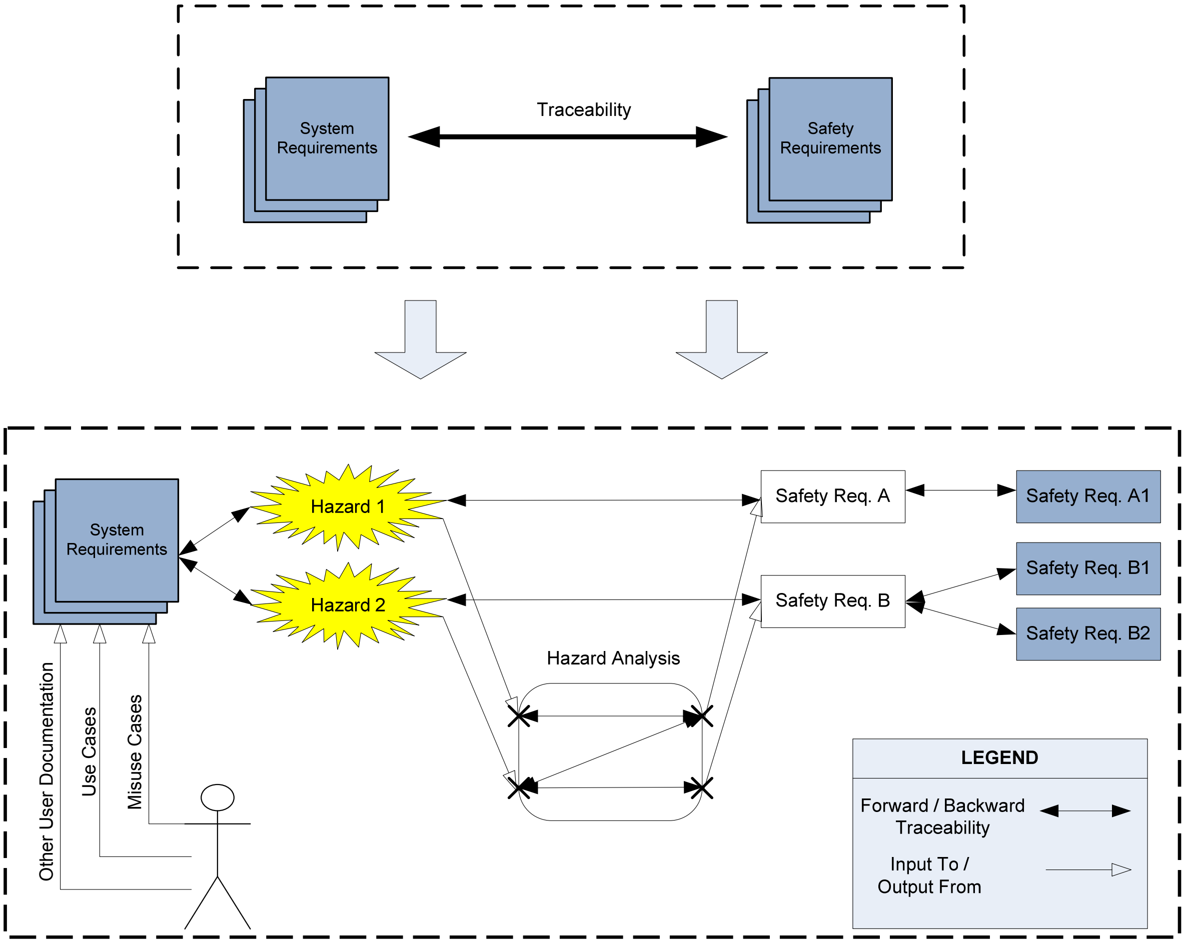


Figure 5. Traceability of Safety Requirements

Weaver describes another aspect of arguing the safety of safety-critical software—Requirements Satisfaction. He states:

> Demonstrating Requirements Satisfaction is based upon showing that the behavior of the components of the system, i.e., hardware, software and other (e.g., human interaction) is acceptable with respect to the system level hazards. [15]

Per Weaver's definition, requirements satisfaction also contributes to the validation of safety requirements when considering "*building the right safety product.*" It also has an element of verification—ensuring that the requirements themselves are satisfied—but based on Weaver's [15] description and subsequent analysis requirements satisfaction is primarily concerned with the behavior of the system against the identified hazards (i.e., the right safety product). Weaver has effectively separated hazard identification and hazard analysis into requirements validation and requirements satisfaction, respectively. However, both hazard identification and hazard analysis have validation aspects. Valid software safety requirements are derived from valid system hazards, therefore the hazard analysis process and products require validation measurements.

One tool for identifying system hazards is the *misuse case*. A misuse case is similar to a use case but it indicates possible misuse of the system, rather than normal, expected, operation by users. Another element of validity of identified hazards is their identification through misuse cases. All hazards identified through, or derived from, misuse cases must be documented as part of hazard identification. Conversely, any hazards identified other than through misuse cases should have a corresponding misuse case—this may require an iterative process whereby new misuse cases are created, furthering the safety engineering team's understanding of system operation under hazardous conditions.

Based on the above discussion, validation of software safety requirements through metrics, from the perspective of the safety engineering team and stakeholders, can be broken into three areas of concern:

- Hazard Identification.
- Hazard Analysis.
- Software Safety Requirements Traceability.

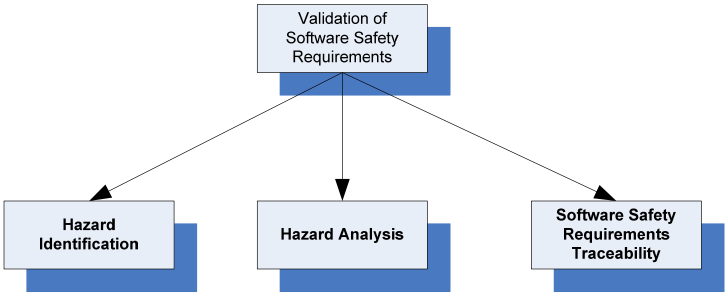

Validation of Software Safety Requirements
Hazard Identification
Hazard Analysis
Software Safety Requirements Traceability

Figure 6. Elements of Validation of Software Safety Requirements

As discussed in Chapter II, the traditional concept of validation (ensuring system specifications meet stakeholder requirements and expectations) is not suited for validation of safety requirements. Apart from generic (usually industry specific) safety requirements, stakeholders will often have one requirement concerning safety: the system must be *safe*. This requirement will often be an expectation, rather than conveyed through stakeholder documentation. The above paragraphs have identified the different areas of the safety process where validation plays a significant role. However, Figure 7 requires the addition of safety requirements for it to more clearly explain the validation connection.

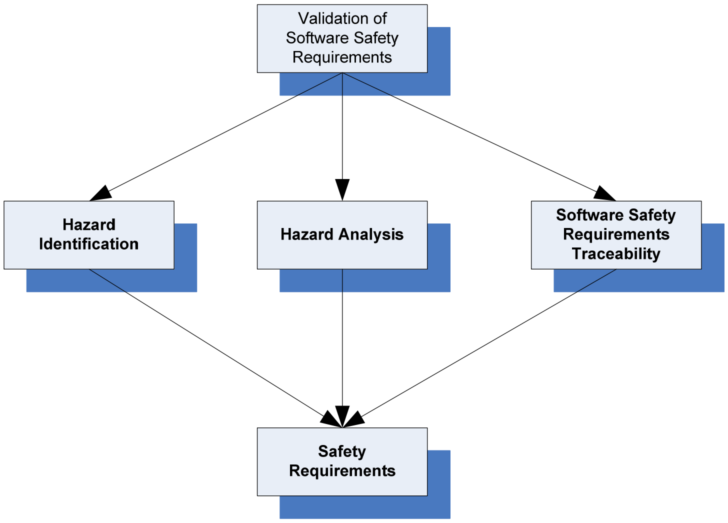

Validation of Software Safety Requirements
Hazard Identification
Hazard Analysis
Software Safety Requirements Traceability
Safety Requirements

Figure 7. Modified Elements of Validation of Software Safety Requirements

The ultimate goal of validation of software safety is to validate the safety requirements. However, because of the lack of stakeholder definition of what constitutes a *safe* system, an alternative validation model to the traditional model is proposed. Validation of safety requires a proxy through which to validate safety requirements. This proxy is the combination of software safety techniques used to derive the software safety requirements. Effectively, the software safety team acts as an advocate for safety on behalf of the stakeholders, determining system safety requirements based on the requirements of the system. Measurements to aid in the validation of safety-critical systems must be derived from the hazard identification, hazard analysis, and requirements traceability artifacts. Figure 8 displays a simplified version of the traditional method of validation and the proposed model for validation of software safety requirements.

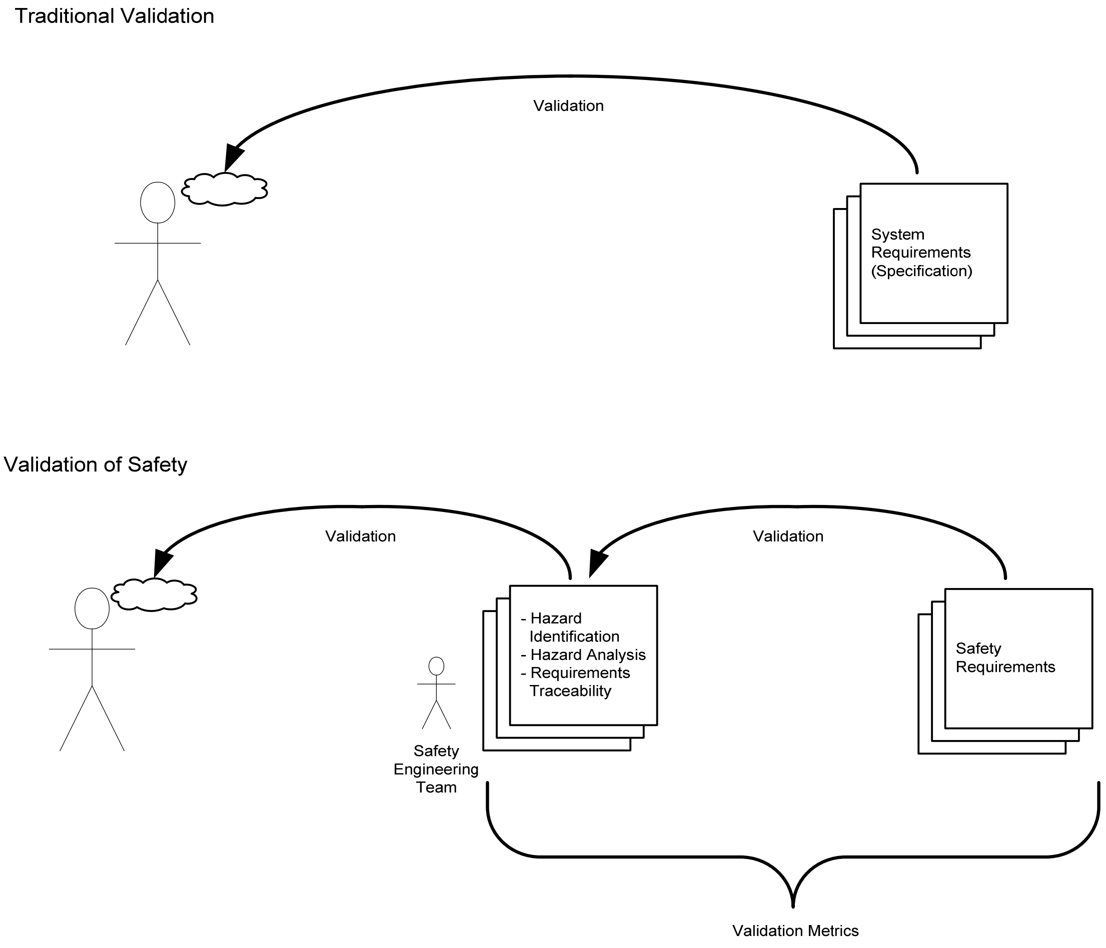


Figure 8. Traditional Validation vs. Validation of Safety

F. GOAL QUESTION METRIC APPROACH

The Goal/Question/Metric (GQM) Approach [17] is a generic framework for defining and organizing software metrics according to organizational objectives. GQM was initially developed by Victor Basili and David Weiss [18] in 1984 for NASA, and since then it has been used by a number of large software engineering corporations [19] (Philips, Siemens, and continued with NASA), resulting in a “de facto” standard for defining measurement frameworks. It employs a top-down definition by focusing on organizational goals and working down to applicable metrics that are used to realize these goals. Although originally developed for defining and evaluating goals in a specific environment, the concept can be employed for nearly any environment where evaluating goals with empirical measurement is required. GQM uses a three-tiered approach for defining appropriate metrics:

- The conceptual level (the Goal) defines measurement goals for product, process, or project.
- The operational level (the Question) defines a set of questions to characterize the object of measurement with respect to a specific quality issue.
- The quantitative level (the Metric) defines a collection of metrics selected to answer the questions in a quantitative way.

Figure 9 presents a graphical model of the three-tiered GQM approach.

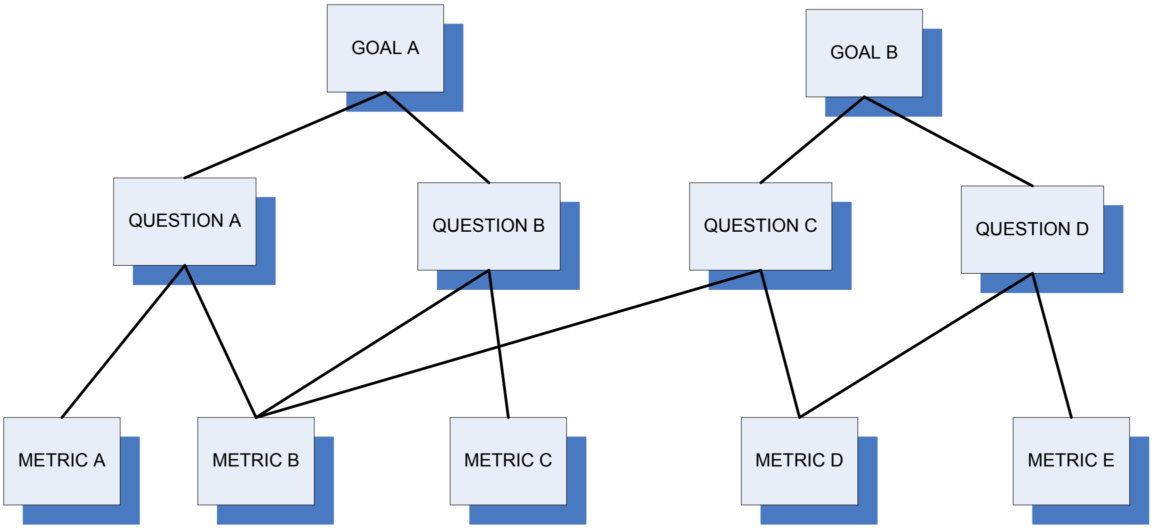


Figure 9. GQM Hierarchy

The resulting metrics are not restricted to any one question. As Figure 9 shows, questions can be answered through a number of metrics.

The organizational aspect of the GQM approach is not restricted to “business” objectives, but rather it is a way of bounding the scope of the goals that can be identified. In the case of this thesis, the organization will be restricted to the safety engineering team.

According to Basili et al. [17], “Measurement is a mechanism for creating a corporate memory and an aid in answering a variety of questions associated with the enactment of any software process.”

Basili et al. [17] also state that, “Measurement, in order to be effective must be:

- Focused on specific goals;
- Applied to all lifecycle products, processes, and resources;
- Interpreted based on characterization and understanding of the organizational context, environment, and goals.”

Basili et al. stress the importance of a top-down methodology with a focus on goals and models. Hence the resulting GQM approach—allowing for goals to be realized through a top-down selection and organization of metrics based on goal-driven questions.

Because of its malleable goal-focused approach, GQM is a perfect candidate for defining a validation metrics framework. As such, the GQM approach will be the cornerstone of the framework that will follow over the next few chapters. Validation is simply another aspect of lifecycle products, processes and resources, therefore lending itself to the application of the GQM approach to select, organize, and define applicable metrics. Goals will be tailored to the validation environment, as discussed in previous sections, with the focus on safety products.

A GQM model is in itself extensible to other GQM models. Goals, questions and metrics from the model can all be utilized by other GQM models throughout the wider organization to satisfy measurement objectives. It is more likely that the metrics themselves will be shared amongst different GQM models.

However, if the goals and questions are applicable to other models, reuse of this information should be capitalized. Therefore, the creation of a validation GQM model should utilize metrics that are already collected (if they aid in answering the questions of the model). Conversely, other GQM models can reap the benefits of already established metrics, such as those under the validation GQM model.

Basili et al. [20] define a set of templates and guidelines for the application of the GQM framework. The conceptual level (goal) template addresses three aspects:

- Purpose.
- Perspective.
- Environment.

The operational level (question) is divided into guidelines for product related questions and for process related questions, each addressing:

- Definition (of product or process).
- Quality perspectives of interest.
- Feedback (relative to the quality perspective).

Guidelines for the quantitative level (metric) are purely dependent on the quantification of the operational level. Basili et al. acknowledge the need for more than one metric to satisfy a question and that both objective (quantitative) and subjective (qualitative) metrics are valid.

The GQM templates and guidelines will be used as a starting point for defining the framework based on the GQM approach. As identified by Basili et al. [20] regarding the templates and guidelines, “they will most likely change over time as our experience grows.”

In this instance, the generalized templates and guidelines will be the initial foundation of the framework. However, specialization will occur to focus on validation of safety-critical software-intensive systems.

Lack of resources often plays havoc with development of software systems. Paucity of resources when applied to the metrics framework is also a possibility. It may be the case that a suitable framework is defined; however, realistically obtaining and analyzing the metrics data is beyond the reach of the safety engineering team due to lack of resources. Therefore, a prioritization scheme should be explored that covers all three levels of the framework (i.e., goals, questions, and metrics). Berander and Jönsson define an Extended GQM Approach [19] to address this issue. The approach utilizes surveys completed by stakeholders leading to the weighting of the goals and questions based on the responses. In the safety-critical environment, prioritization and weighting should also be determined based on such things as hazard levels and residual mishap risk levels. The Extended GQM Approach identified by Berander and Jönsson could be a realistic prioritization method for a validation metrics framework.

G. SOFTWARE DEVELOPMENT LIFECYCLE PROCESS

Software Development Lifecycle Processes (SDLPs) come in many flavors and assortments. Understanding the basic characteristics of each major SDLP is necessary to define a validation metrics framework that is:

a. Independent of the SDLP used, or

b. Tailorable to the SDLP used.

Although there are many other SDLPs used throughout the software engineering community, four major (and currently in use) methods are:

1. Waterfall,
2. V-Model,
3. Spiral, and
4. Iterative and Incremental Development (IID).

These SDLPs can be classified as either sequential or evolutionary. The Waterfall, V-Model, and Spiral SDLPs are all sequential (typically), meaning that each phase in the lifecycle is performed upon completion of the previous phase.

Evolutionary SDLPs (IID) are not necessarily sequential in nature—concurrent engineering is a focus and repetition of the lifecycle is planned for.

The Waterfall model is a sequential development strategy based on the major elements of any system development. It was originally proposed by Winston Royce [21] in 1970 as a method for developing large (or complex) software systems. Although Royce intended for his proposed model to be an iterative approach, this was largely ignored and instead the sequential “waterfall” model was adopted. Regardless of whether an iterative or sequential waterfall model is used, the following characteristics are typical:

- Big Design Up Front (BDUF). The model requires a thorough understanding of requirements and large up front effort to ensure expectations are met.
- Does not deal well with risk. Risk is not a focus of the process. Risky elements of the project may be afforded more resources, but there is no avenue for minimizing risk by breaking the large risk down into smaller more manageable risks.
- Does not deal well with complexity. The process is typically monolithic in nature and complex systems become unwieldy and very difficult to understand under such a process.
- Simplest and most efficient method for small projects. Resources are not wasted as the process includes only the “bare-essentials” to get the job done.
- Validation is a product of user acceptance testing, falling under the *Testing* or *Operations* activities.

The waterfall model, as described by Royce [21], is shown in Figure 10.

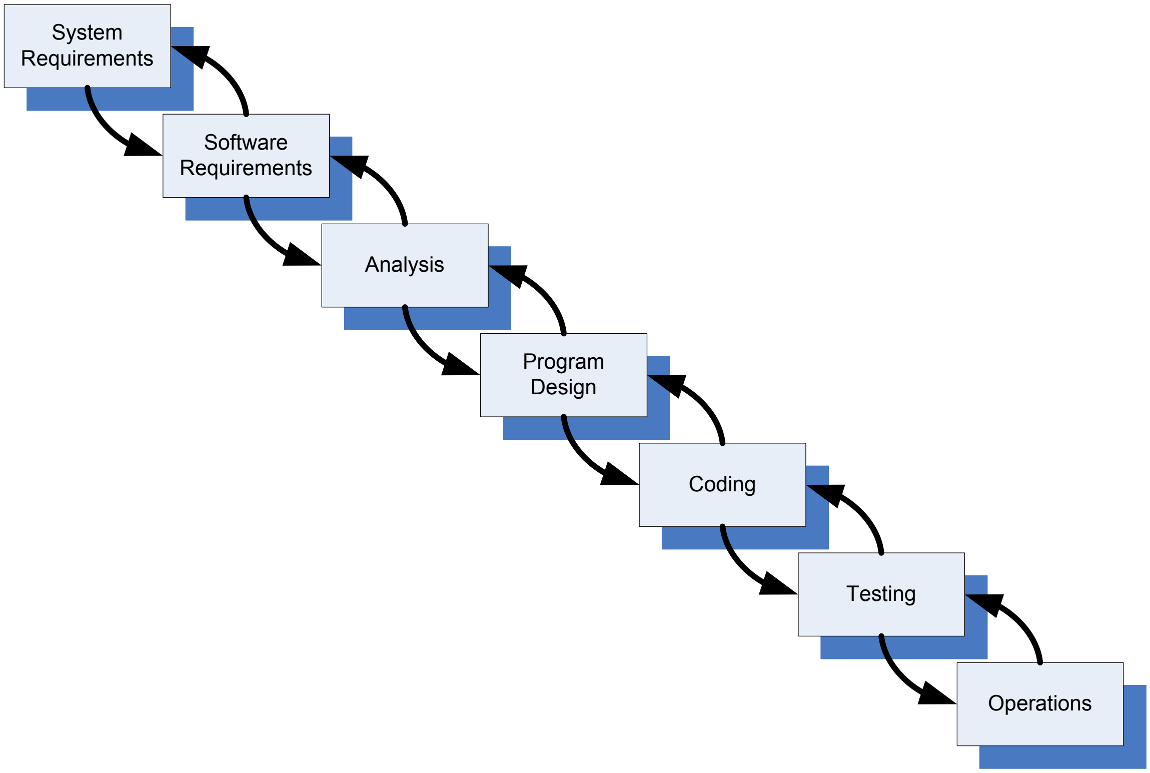


Figure 10. The Waterfall SDLP

The V-Model is an extension, or modification, of the Waterfall SDLP. Instead of relying on a purely hierarchical sequential structure, it realizes that verification occurs (or should occur) at more intervals than just during the *Testing* phase. It is still a sequential method—development occurs on the way down the left side of the V, and verification occurs against each element on the right side of the V. Testing is still performed sequentially from the bottom up, and validation effectively occurs as verification tests are carried out. The V-Model has the following characteristics:

- Incorporates all the characteristics as the Waterfall model.
- Verification occurs at multiple levels. Verification is the focus of this model. It ensures that each level of development, from requirements analysis through to implemented code, is verified against applicable requirements.

- Validation is obtained through verification activities. Although not designed to occur this way, any verification activities will have some influence on validation by revelation of the system capabilities to stakeholders through testing. This validation technique builds upon the Waterfall model by breaking testing into identified elements and testing from the lowest level to the highest requirements.

Figure 11 shows the V-Model SDLP.

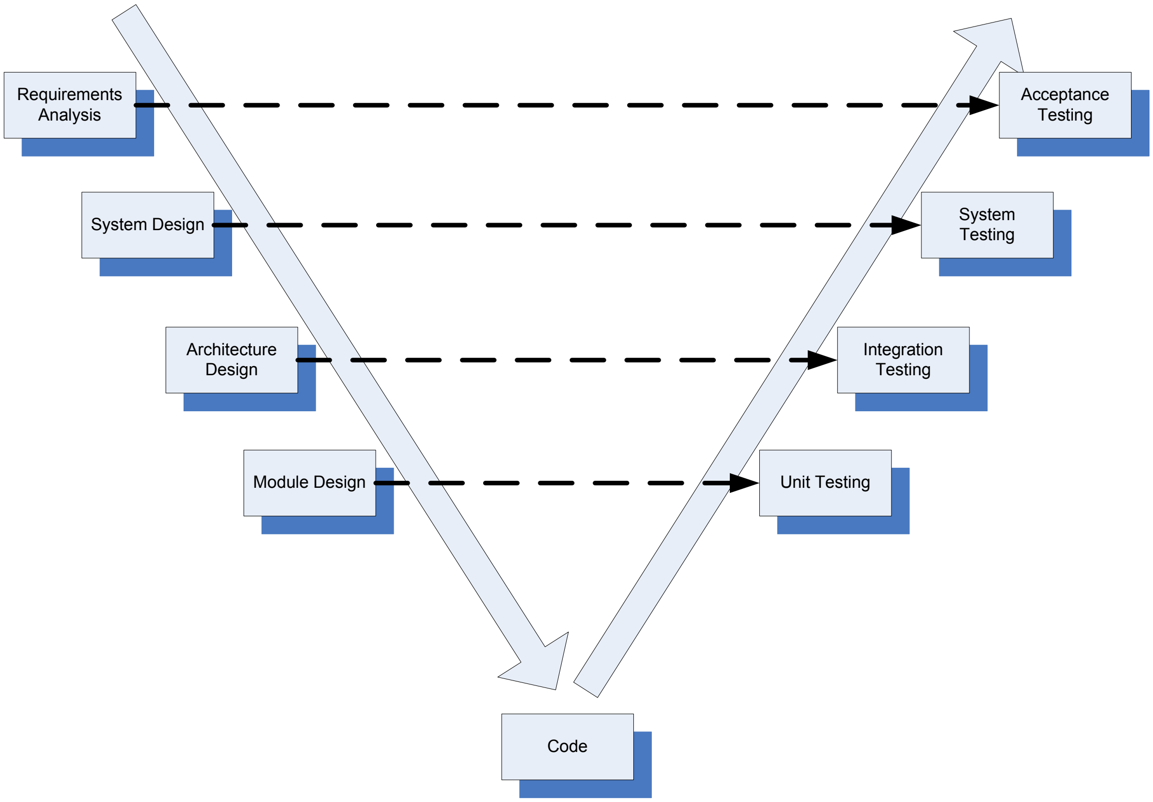


Figure 11. The V-Model SDLP

The Spiral Model was introduced by Boehm in 1988 to address many of the shortfalls experienced with the typical Waterfall SDLP. Boehm [22] proposed a *risk-driven* approach to software development, consisting of a series of *spirals* with clear objectives, alternate means of implementation, and constraints imposed. Essentially the model is the Waterfall model with a focus on risk analysis. Each spiral ends with a *prototype* product, dependent on the progression through the development process, which is then verified through simulations, models, or benchmarks. The Spiral Model does enhance the Waterfall Model to a large degree; however, it also carries over many of the same characteristics. The following are the characteristics of the Spiral Model:

- The BDUF concept is broken down. Because the Spiral Model has a focus on progressive *spirals* of requirements definition and analysis, the BDUF concept is broken into more manageable chunks. Although this is an advantage, detailed specification is still a prerequisite to implementation.
- Design alternatives through reuse or reworks are incorporated. This allows for analysis of possible reuse candidates, or if mistakes are made (usually in understanding users expectations—validation) rework can be performed.
- Risk analysis as a mandated step results in careful planning.
- Validation is a more realistic concept through the development of prototypes (whether they be requirements specification, detailed requirements, module design, etc.). This allows for language translation to occur through regular reviews and testing at the end of each spiral; however, it does not specifically address validation.

The Spiral Model is shown in Figure 12.

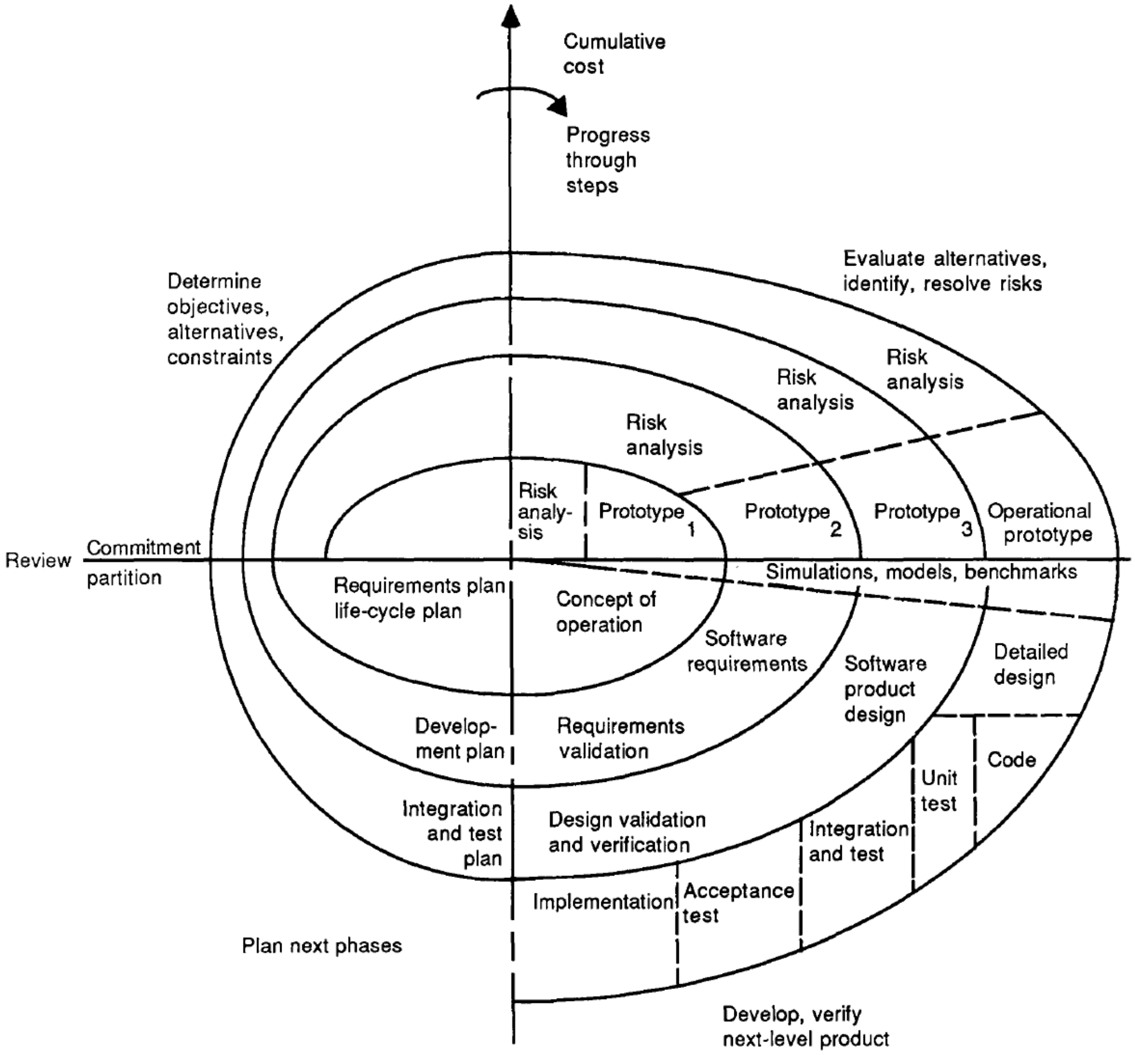


Figure 12. The Spiral SDLP [From [22]]

The Iterative and Incremental Development (IID) model is the most recent SDLP of the four. IID is an evolutionary and progressive method. According to Larman [23]:

> The iterative lifecycle is based on the successive enlargement of a system through multiple iterations, with cyclic feedback and adaption as core drivers to converge upon a suitable system. The system grows incrementally over time, iteration by iteration.

IID aims to address the issue of complexity. As systems become increasingly complex, sequential development methods (all previously mentioned

SDLPs) become exceedingly difficult to implement. IID allows for the development effort to be broken into more manageable portions of logic and customer feedback is welcomed throughout the process. The following are characteristics of the IID SDLP:

- The BDUF concept is non-existent. Fully defining all system capabilities is often challenging or near impossible for many large systems. IID embraces this problem, and therefore does not mandate complete specification prior to development.
- Risks are identified, or become self evident, incrementally. Rather than discovering integration issues late in development, or safety hazards upon deployment, risks are identified as a more thorough understanding of the system is gained through each increment of design and development.
- Validation is a core focus of IID. Although typically not explicitly identified as validation, user/customer feedback as the system is incrementally delivered allows for expectations to be more closely met.
- Requirements instability is a reality. Requirements are often not stable until the final iteration of development. This in itself can become subjective, e.g., when is the final iteration if requirements are not stable?
- "*Scope-creep*" is a very real threat. IID requires sufficient bounding over its complete lifecycle to prevent excessive requirements change (this characteristic closely ties with the previous).
- Difficult to contractually manage. Due to the fact that IID does not mandate complete specification prior to design and development (rather it encourages exploration of system requirements and understanding), it can often be difficult to implement for large, complex systems under contract.

Figure 13 shows the IID SDLP. Each vertical line represents an incremental build of software (although the headings detail Inception, Elaboration, Construction, and Transition, each of these activities are undertaken for each iteration). They are, however, iteratively revisited through Business Modeling, Requirements, Analysis and Design, Implementation, Test, and Deployment throughout the entire SDLP.

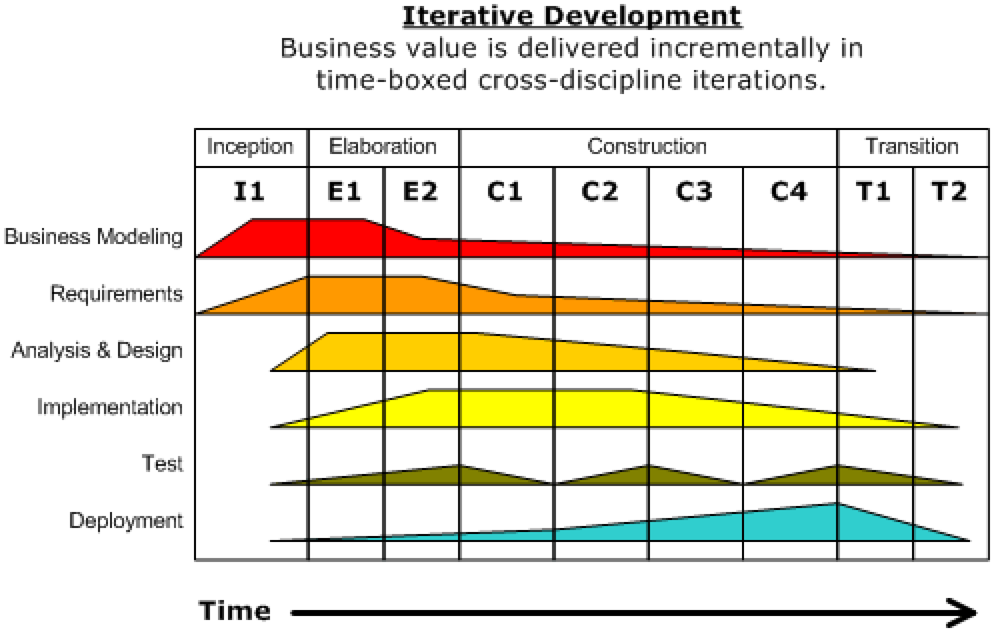


Figure 13. The Iterative and Incremental Development (IID) SDLP

The above SDLPs are typical of those currently used in the software engineering discipline. Their characteristics differ significantly, and therefore consideration of these characteristics is required for developing a validation metrics framework. As the framework is developed throughout the following chapter, these SDLPs will influence the possible solution(s). To cater to each SDLP, the framework will either be tailorable to the specific SDLP or be independent of the SDLP.

When considering the above SDLPs from the safety engineering team's perspective, there are also large impacts on the way safety is achieved. Much of the software safety process (as defined by Leveson [14]) relies on definition and design of the subject system to relate hazardous conditions to software, hence defining what is safety-critical software and what is not. Therefore, depending on the SDLP used, the framework will behave differently due to the fact that development is carried out differently. For the SDLPs that focus on the BDUF principle, complete design data will be available sooner (theoretically) than an evolutionary SDLP. In the case of the IID SDLP, requirements and design artifacts will be incrementally produced. Thus, identification of new safety-critical software, and most likely new hazards, will occur throughout the process. This will also occur on many BDUF SDLPs, though not to the same order of magnitude that would be expected of an evolutionary type of development. This is another aspect that the Validation Metrics Framework needs to account for when considering the type of SDLP used.

H. FRAMEWORK OBJECTIVES

Safety-critical software-intensive systems are subjected to the highest levels of scrutiny and quality assurance in software engineering. A holistic, systematic approach to the validation of these systems is required. Currently none exist. Validation metrics can be used to ensure user and stakeholder requirements and expectations are closely matched. However, due to the possibly vast number of domains that users and stakeholders can represent, there is no "*silver bullet*" that will be applicable at all levels within all domains. Therefore, the intent of this thesis is to establish a framework for validation metrics applicable to safety engineering teams. The safety engineering team is typically responsible for constructing a safety argument supported by a collection of evidence on the application of safety engineering for a system's development. Validation metrics data will aid in forming this argument as it is presented to stakeholders concerned with determining the validity of system safety requirements—addressing the key elements of *Hazard Identification, Hazard Analysis* and *Safety Requirements Traceability*.

The framework must cater for the unique aspects of software safety. Reliability is not a key indicator of safety and therefore will not be the focus of software safety. The software safety process, as described by Leveson [14], is to be central to the development of the Validation Metrics Framework. Therefore, measurement data will arise from products associated with the software safety process.

The Validation Metrics Framework must be able to identify existing metrics (should there be any that are suitable) that may be used in the validation context (or at least define their characteristics) and define the characteristics of those metrics that do not exist (to aid in their development). The key focus of the framework will be that of providing the users of the framework with actionable data (from metrics) based on the requirements and expectations of stakeholders with a focus on system safety. The safety engineering team will develop their own understanding of system safety based on high-level user documents from the users and stakeholders (which can consist of any user or stakeholder document describing how they expect to use the system), system specification, and subsequent documentation by the development team.

Traceability of requirements is one of the key concerns of modern systems engineering, as has top-down approaches to system design becoming a common practice in all engineering disciplines. The framework for validation metrics should be congruent with requirements traceability and top-down design. A goal-driven approach will ensure that every metric has a purpose, reducing wasted collection of data, and ensuring that the rationale behind measurements is justifiable. Traceability from goals to metrics and metrics to goals will be a key objective of the framework. For this thesis, the GQM approach will be the cornerstone of the framework, allowing for development of metrics in a controlled and purposeful approach for validating the safety of a system.

The validation activity is not restricted to simply the safety engineering team. Although the focus of this thesis is safety, the framework must be sufficiently open to other aspects of validation and metrics gathering on a system. Therefore, prioritization of validation metrics within the framework should be possible, and also prioritization through external influences. Metrics may be gathered by many different stakeholders of the system. Thus, the Validation Metrics Framework should be able to become part of a hierarchically prioritized overarching framework (i.e., the sum of the system concerns), influencing overall prioritization of measurements.

The framework must cater for major SDLPs (identified through the four described previously, however characterized as sequential or evolutionary) characteristics either through tailoring to suit each specific instance, or by independence from the SDLPs.

In summary, the Validation Metrics Framework objectives are:

- Scope is restricted to validation metrics applicable to the safety engineering team in the sense that they will be the users of the framework. The resulting data aiding in validation of safety will be presented to users and stakeholders through safety arguments/cases.
- Cater for the unique aspects of the software safety process (based on products and artifacts).
- Identify existing metrics (or at least characteristics required) and define characteristics of non-existent metrics (to aid in their development).
- Utilize a top-down goal-driven approach (GQM) to ensure relevance of data, identify the applicable environment/audience, and provide traceability.
- Explore the possibility of allowing for prioritization of measurements based on safety aspects (e.g., hazard levels and residual mishap risk) and be amenable to prioritization influence from overarching metrics frameworks.
- Cater for a range of SDLPs (focusing on the four described previously) by either tailorability or independence from the SDLPs.

These objectives will flow through as inputs to the development of the Validation Metrics Framework in the following chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DEVELOPMENT OF THE FRAMEWORK

A. INTRODUCTION

The framework objectives detailed in Chapter III serve as the primary focus for the development of a validation metrics framework. This chapter will further analyze specific attributes of the framework to describe how it will be composed, as well as the process for application. The following chapter will demonstrate application of the framework through a case study.

B. FRAMEWORK GOAL STRUCTURE

The GQM approach as defined by Basili et al. [17] proposes a single level of goals in the GQM hierarchy. However, to aid in defining clear goals for the Validation Metrics Framework, it is prudent to determine an overarching framework goal (in the format that Basili and Rombach [20] provide) prior to these lower-level goals. By developing a hierarchy of goals from the highest level, sufficient depth will result so that the formulation of subsequent questions will be an achievable task. This concept will also ensure that a truly top-down validation focus is maintained throughout. Figure 14 demonstrates the proposed architecture.

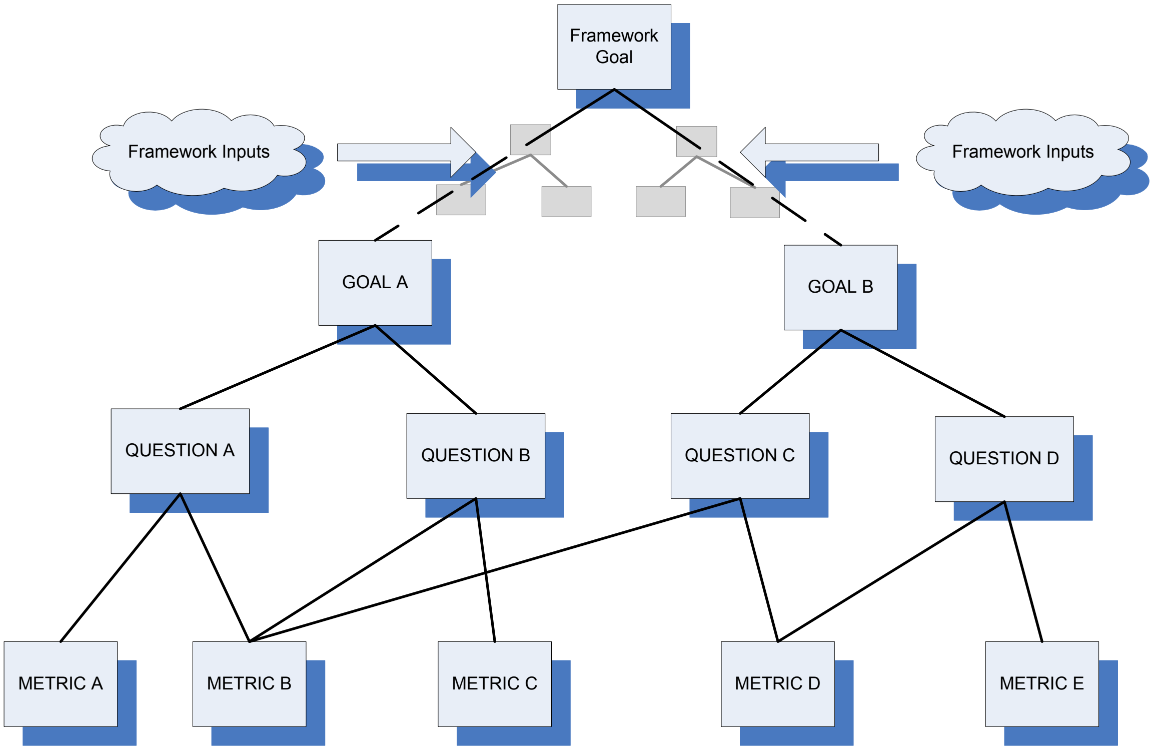


Figure 14. Validation Metrics Framework Goal Hierarchy

The Framework Goal is the highest level goal defined. For the purposes of this thesis, it restricts application as per the objectives of Chapter III. That is, it focuses on the safety engineering team as the user of the framework within a validation environment. The dashed lines between the Framework Goal and goals A and B indicate that there may be intermediate levels of goal definition as goals are broken up in a top-down fashion to a suitable level for formulating applicable questions. Influencing these intermediate goals (or even the resultant low-level goals if no intermediate goals exist) are framework inputs. These inputs consist of many of the objectives mentioned in Chapter III. The following section will provide detail on framework inputs.

Using Basili and Rombach's [20] templates for goal definition, the Framework Goal definition is:

- **Purpose:** To measure the products of the software safety process throughout development and implementation in order to aid in validating software safety requirements.

- **Perspective:** Examine the metrics from the safety engineering team’s point of view, with a focus on validating software safety requirements by proxy—in accordance with the proposed model for validation of software safety requirements.
- **Environment:** The system has safety-critical elements that will be bound by the software safety process.

This Framework Goal will carry over to any application of the Validation Metrics Framework within the scope of this thesis, i.e., use by the safety engineering team for safety-critical software-intensive systems, as goals are identified in a top-down fashion, ensuring context is maintained. Given another area of application, the Framework Goal could be changed to suit.

Determining subsequent levels of goals is dependent on the framework inputs as discussed below. Following identification of these inputs, guidance will be given on the formulation of the remaining goals and questions.

C. FRAMEWORK INPUTS

The inputs to the framework are of significant importance. These inputs will influence how the lower-level goals are defined. Inputs to the framework will be identified by their influence on the framework goals; that is, for an element to be considered an input, it must have some relevance to the goals. As stated earlier, goals are defined in terms of purpose, perspective, and environment. These three aspects will be the focus of any input to the framework.

1. SDLP Input

The SDLP chosen will have a significant affect on the types of goals that can be formulated, particularly when considering the environment aspect of the goal. Following the Framework Goal, the SDLP needs to be noted as an environment factor for the subsequent goals to clarify what is realistic, or even possible. For example, a goal of maintaining safety requirements stability to within 5% (i.e., change of number of safety requirements over the lifecycle, or similar) while using an evolutionary SDLP is not a realistic goal since requirements in an evolutionary SDLP are inherently unstable.

Another factor that needs to be considered is when safety products (the results of the system software safety process) will be available for measurement and how the chosen SDLP affects these measurements. The system software safety process described by Leveson [14] is portrayed in a "Waterfall" fashion (however, it does not imply that the process must be carried out sequentially). Therefore, when employed in an SDLP other than sequential, care must be taken to realize that the process itself will be iterative in nature. Safety requirements will change as the system matures either iteratively, incrementally, or both. New safety-critical software components will be identified as the design matures in each pass through the evolutionary process. Even in the V-model SDLP, verification of safety requirements will occur at different stages.

The SDLP framework input will influence the environment factor of goals identified. This will shape subsequent goals, questions, and metrics to ensure that data being gathered is tailored to the SDLP chosen, and that data presented to the framework user is applicable to their situation. The following SDLP characteristics need to be considered as environmental factors when developing intermediary goals:

- SDLP method. Is it evolutionary or sequential? This will be the major characteristic that will influence many of the following characteristics.
- Safety stability. Will the system software safety process products be continuously changing because of the type of SDLP? This will be a key influencing factor on metrics gathered.
- Current validation methods. How is validation of the system currently carried out in the chosen SDLP? An SDLP that supports operator and customer feedback throughout will provide for more opportunity to clarify validation of safety through metrics. One that

does not support this interaction will need to consider the appropriate goals for ensuring that there is sufficient feedback from the user community.

- Safety process execution. How is the safety process carried out in the chosen SDLP? Will all safety products be available up front, or will they become available as the system matures? Hazards, for example, identified early on in the project may become redundant as the system matures, or there may be new ones identified. The goals of the framework need to be amenable to these changes.

Generally, the SDLP chosen will not directly determine the goals identified for the generic Validation Metric Framework. It will be an influencing factor on the environment component of the goals; however, this is simply a contextual impact. Interpretation of any actionable metric data will need to be undertaken in the context of the SDLP.

2. Safety Input

The software safety process used will influence framework goals. This process will be the driving factor behind what products will be available for measurement throughout development.

Safety products that are available for measurement will vary depending on the exact software safety process used. A safety product reference list should be compiled to ensure that all safety products are known and can be planned for inclusion as input to the framework. This list will need to be identified for each application of the framework and will serve as a reference list for the specific goals, questions and metrics. It will serve as a “sanity” check for what is obviously possible and not possible, according to the types of products available, but also serve the same purpose for those goals that may not be so obviously possible or not. For example, if a goal is identified, question derived, but then no product available to obtain a metric from, the goal either needs to be revised or revoked. The safety product reference list is a simple method for identifying appropriate product metrics.

Some typical system software safety products are:

- Preliminary Hazard List (PHL),
- Preliminary Hazard Analysis (PHA),
- Hazard analysis reports: Fault Tree Analysis (FTA), Failure Modes Effects and Criticality Analysis (FMECA), Event Tree Analysis (ETA), Hazard and Operability Studies (HAZOP), etc.,
- Independent Safety Audit Reports,
- Safety Requirements Specification,
- System Safety Requirements Traceability Matrix (SSRTM), and
- System Software Safety Requirements Traceability Matrix (SSSRTM).

An exhaustive list is not provided, as each project will define their own version of the system safety process, therefore tailoring the products that will result. The above list serves as an example of possible products that may comprise the safety product reference list.

3. Stakeholder Feedback

To ensure a truly effective and efficient Validation Metrics Framework, the framework needs to allow for inputs of stakeholder feedback. Primarily this will be through clarification of high-level user requirements, but could also be as a direct result of the metrics being presented. Initially the metrics would be available to the safety engineering team, but this same information would also be presented to necessary stakeholders, particularly the operators and owners. The results of the metrics may influence composition of goals, or directly the goals themselves. It may also have an impact on prioritization. Therefore, any stakeholder feedback needs to be combined as part of the framework inputs.

The identified framework inputs are shown graphically in Figure 15.

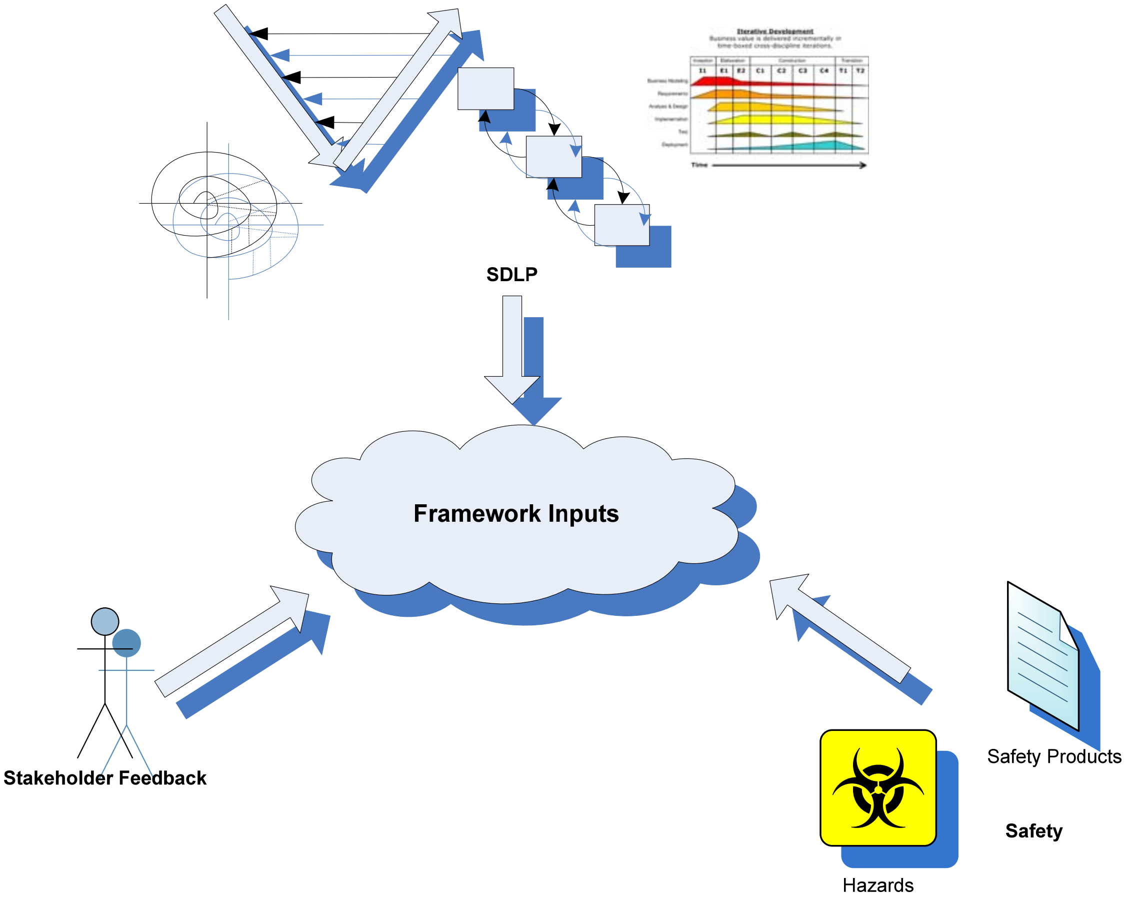


Figure 15. Framework Inputs

D. IDENTIFYING GOALS, QUESTIONS AND METRICS

Following on from the Framework Goal definition, intermediary goals must be identified to a sufficient level as to enable the creation of questions. Because of the hierarchical nature of the framework, intermediary goals will be influenced by the Framework Goal and framework inputs.

Goals may or may not continue to be described through *purpose, perspective, and environment* as proposed by Basili and Rombach. This will depend on the clarity of the goals in a few words, dependent on the personnel involved. Although the Framework Goal is given in this format, the following derived goals (for this thesis) will simply be presented in a few words. Further clarification can be given if required. The use of a hierarchical top-down goal derivation structure will aid in reducing complexity of defining goals, and ensure that the scope and focus for metric identification is maintained.

Each intermediary goal must measure products from all stages of the system software safety process including:

- Conceptual development.
- System design.
- Full-scale development.
- System production and deployment.
- System operation.

The framework results, or measurements, will change, develop, and mature as the system itself matures. Depending on the SDLP used, these will either be slight changes or continual development. Although a generic framework will initially be given, it is expected that goals may be added, removed, or improved; framework inputs will be added, removed, or improved; or stakeholder feedback will influence the goal structure.

To aid in the goal derivation process, the Goal Structuring Notation (GSN) method proposed by Weaver et al. [24] will be used. Weaver et al. [24] state that, “The Goal Structuring Notation (GSN)… is a graphical notation for constructing complex safety arguments for safety cases.”

Although specifically designed for constructing safety arguments, the method also lends itself to rational development of hierarchical goal structures while ensuring that context, justification, assumptions, strategies, and solutions are all included.

Briefly, the GSN method consists of elements shown in Figure 16.

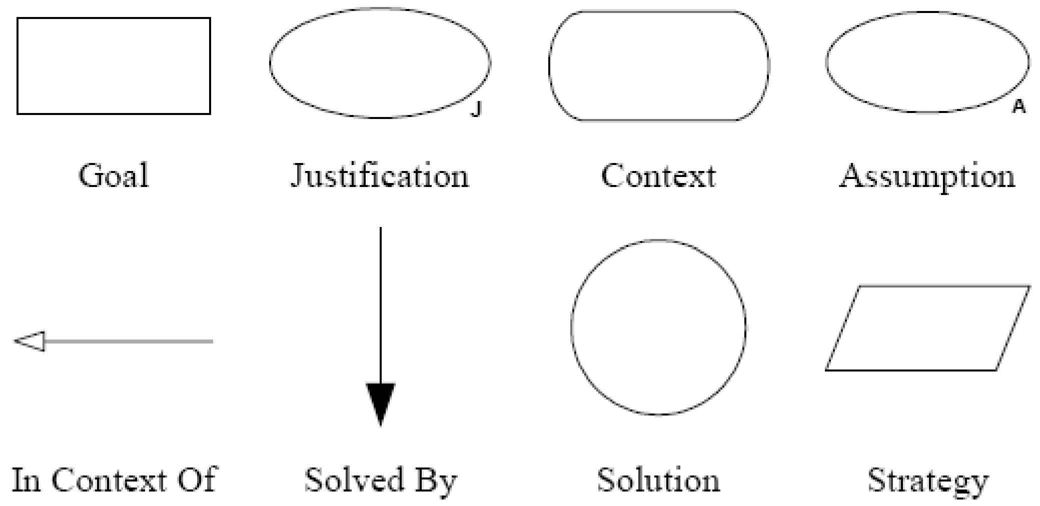


Figure 16. Principle Elements of GSN [From [24]]

Using these elements, the goal hierarchy of the Validation Metric Framework can be composed in a rational and justified manner, whilst still maintaining a top-down approach within the appropriate context. An example of how the GSN method could be used to determine a goal structure is shown in Figure 17.

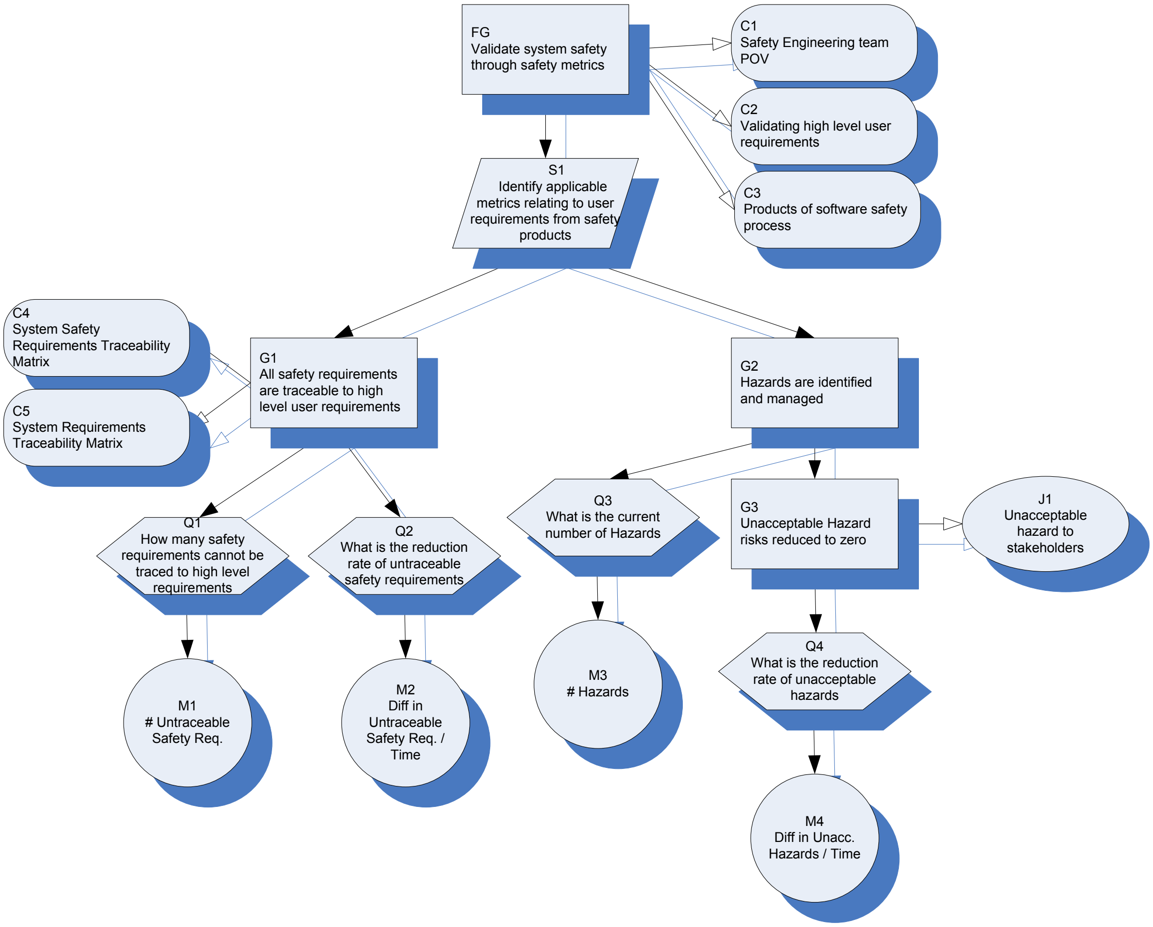

FG
Validate system safety through safety metrics
C1
Safety Engineering team POV
C2
Validating high level user requirements
C3
Products of software safety process
S1
Identify applicable metrics relating to user requirements from safety products
C4
System Safety Requirements Traceability Matrix
C5
System Requirements Traceability Matrix
G1
All safety requirements are traceable to high level user requirements
G2
Hazards are identified and managed
Q1
How many safety requirements cannot be traced to high level requirements
Q2
What is the reduction rate of untraceable safety requirements
Q3
What is the current number of Hazards
G3
Unacceptable Hazard risks reduced to zero
J1
Unacceptable hazard to stakeholders
Q4
What is the reduction rate of unacceptable hazards
M1
# Untraceable Safety Req.
M2
Diff in Untraceable Safety Req. / Time
M3
# Hazards
M4
Diff in Unacc. Hazards / Time

Figure 17. GSN Example

In the above example, GSN is combined with the GQM framework creating a hybrid solution. GSN is used to develop and convey the goal structure, taking into account any necessary *purpose, perspective, and environment* attributes that benefit from being displayed. These are indicated through the goal itself and the context notation. As with the originally proposed Validation Metrics Framework depicted in Figure 14, goals can be decomposed into subsequent lower-level goals to allow for easier derivation and more precise definition of the following question. Not shown in the GSN notation proposed by Weaver et al. is the inclusion of questions. Since the Validation Metrics Framework specifically treats metrics, applying the GQM process allows questions to be derived from the goals. Questions are indicated by hexagons in Figure 17. In the GSN diagram solutions are shown by circles. In the case of the Validation Metrics Framework, the solutions are in fact metrics.

In the above example, the following text notation is used:

- FG: *Framework Goal*
- C1, C2, C*n*: *Context*
- S1: *Strategy*
- G1, G2, G*n*: *Goals*
- Q1, Q2, Q*n*: *Questions*
- J1: *Justification*
- M1, M2, M*n*: *Metric.*

In many cases the visual representation will not be sufficient to clearly articulate all aspects of the goals, context, justification, strategy, or questions. Therefore, accompanying the visual GSN/GQM-based representation of the framework should be documentation expanding on these aspects. Specifically, where a goal is identified, in keeping with the *purpose, perspective, and environment* attributes, only those attributes that are required to ensure that context is clearly understood in the visual representation will be shown. Otherwise, the attributes (should they be required for clarity) will be noted in the documentation such that it is accessible for reference. The primary reason for this is to reduce clutter and maintain simplicity in the GSN diagram.

As discussed in the previous chapter, the three elements of validation of software safety will be: *Hazard Identification, Hazard Analysis, and Software Safety Requirements Traceability*. Therefore, these elements need to be incorporated into the goals of the framework. Utilizing GSN, the most applicable place for these elements will be to reflect them in *strategies* and/or *goals*.

Based on the three elements of validation of software safety, the GQM framework, and GSN—Figure 18 shows the top level goal structure for the Validation Metrics Framework.

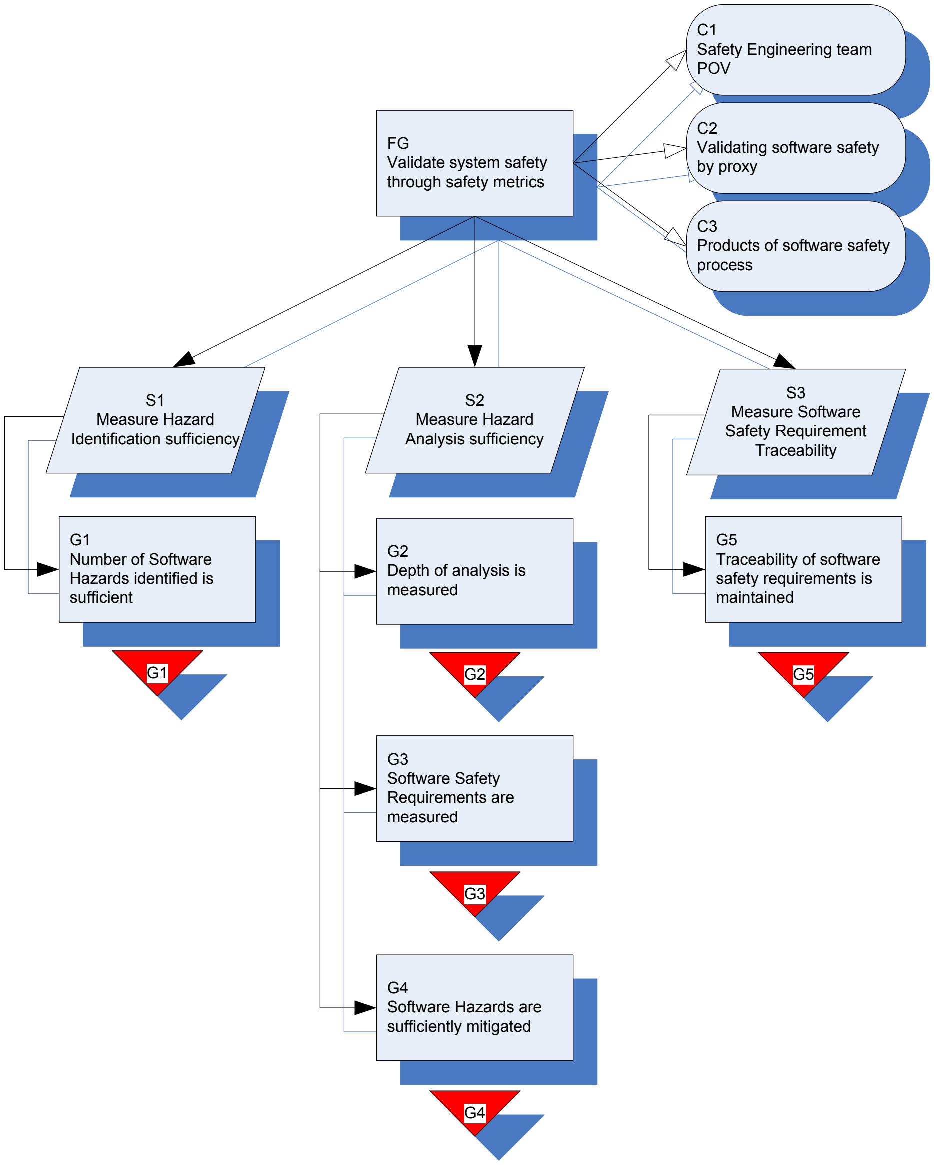

C1
Safety Engineering team POV
C2
Validating software safety by proxy
C3
Products of software safety process
FG
Validate system safety through safety metrics
S1
Measure Hazard Identification sufficiency
S2
Measure Hazard Analysis sufficiency
S3
Measure Software Safety Requirement Traceability
G1
Number of Software Hazards identified is sufficient
G2
Depth of analysis is measured
G5
Traceability of software safety requirements is maintained
G1
G2
G5
G3
Software Safety Requirements are measured
G3
G4
Software Hazards are sufficiently mitigated
G4

Figure 18. Framework Top-Level Goal Structure

The top-level goals will flow into either intermediary goals or questions as the framework continues to be developed. Each goal has a red arrow with a corresponding identifier that will be used in the following GSN structures. Each of the identified goals have been identified as generic to any safety-critical software-intensive system, but are specifically based on the understanding of validation of software safety developed thus far. The following paragraphs will provide more explanation on each goal.

G1: Number of software hazards identified is sufficient. This goal directly addresses the *Hazard Identification* element of validation of software safety requirements. The aim of this goal is to ensure that a sufficient number of software hazards are identified for a particular system. If the number of identified hazards is outside of a pre-determined upper or lower boundary, it can indicate that the Hazard Identification is not sufficient, therefore resulting in potential for invalid or incomplete software safety requirements.

G2: Depth of analysis is measured. This goal partially addresses the *Hazard Analysis* element of validation of software safety. Depth of analysis is associated with the inherent risk associated with identified software hazards. By ensuring that the hazard analysis is performed to a sufficient level of depth (second- and third-order causal factors) for appropriate software hazards, the validity of the analysis is strengthened.

G3: Software safety requirements are measured. This goal also partially addresses the *Hazard Analysis* element of validation of software safety. The result of hazard analysis is that requirements are identified to sufficiently mitigate the software hazards. By measuring the number of software safety requirements against a pre-determined model, the sufficiency of hazard analysis (and therefore validity of subsequent requirements), can be obtained.

G4: Software hazards are sufficiently mitigated. This goal also partially addresses the *Hazard Analysis* element of validation of software safety. Measurements that show hazards have associated mitigating safety requirements will aid to strengthen the validation case.

G5: Traceability of software safety requirements is maintained. This goal directly addresses the *Software Safety Requirements Traceability* element of validation of software safety. Ensuring that all safety requirements can be traced through hazard analysis to hazards, which are then traceable to system requirements, aids in ensuring valid software safety requirements.

Following on from the top-level goal structure, questions and metrics are then identified to fulfill each goal. The following GSN structures show the lower half of the Validation Metrics Framework.

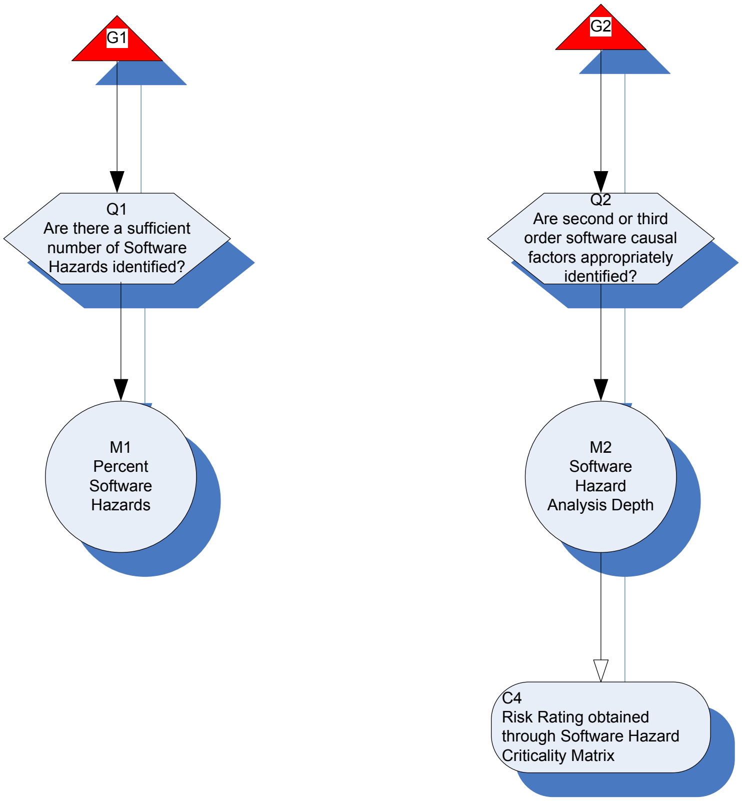

G1
Q1
Are there a sufficient number of Software Hazards identified?
M1
Percent Software Hazards
G2
Q2
Are second or third order software causal factors appropriately identified?
M2
Software Hazard Analysis Depth
C4
Risk Rating obtained through Software Hazard Criticality Matrix

Figure 19. Framework Lower-Half Part 1

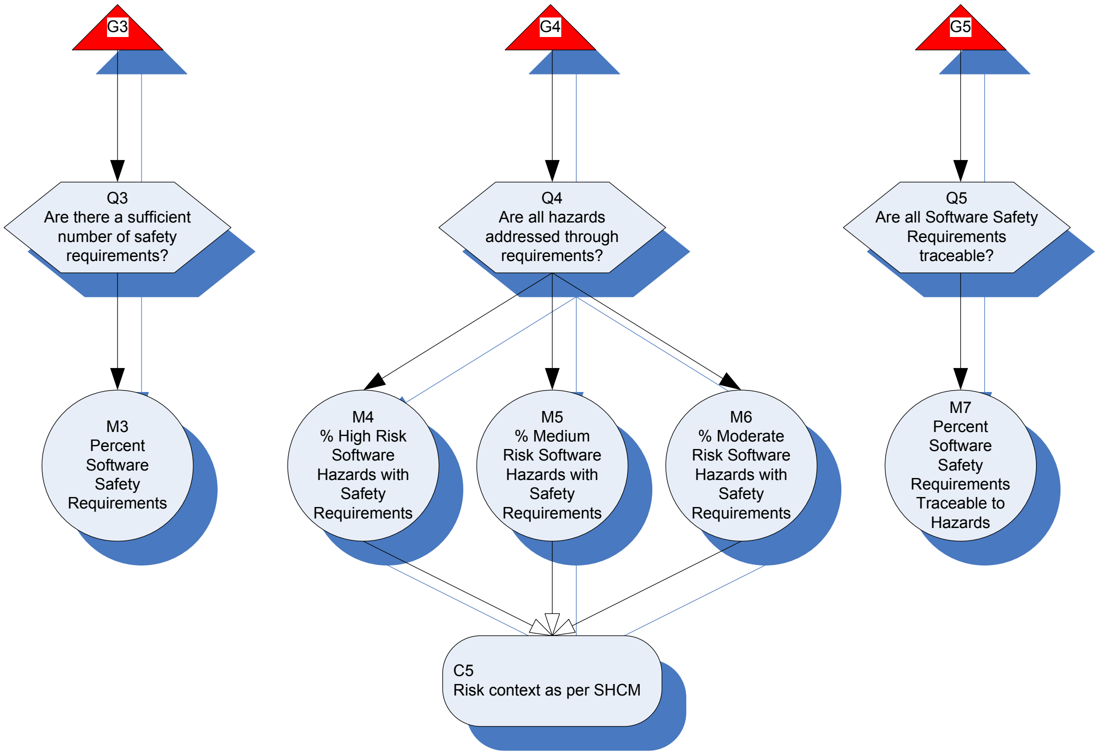

G3
G4
G5
Q3
Are there a sufficient number of safety requirements?
Q4
Are all hazards addressed through requirements?
Q5
Are all Software Safety Requirements traceable?
M3
Percent Software Safety Requirements
M4
% High Risk Software Hazards with Safety Requirements
M5
% Medium Risk Software Hazards with Safety Requirements
M6
% Moderate Risk Software Hazards with Safety Requirements
M7
Percent Software Safety Requirements Traceable to Hazards
C5
Risk context as per SHCM

Figure 20. Framework Lower-Half Part 2

The derivation of the lower-half of the Validation Metrics Framework shown in Figure 19 and Figure 20 result in a generic set of validation metrics for use on any safety-critical software-intensive system. Some of the above metrics are self explanatory while others are not. The following paragraphs detail each of the validation metrics.

M1[6]: Percent Software Hazards. M1 (PSH) is a direct indicator of the sufficiency of *Hazard Identification*. By comparing the number of software hazards identified against historical data, it indicates the validity of the software safety requirements through identified hazards. Interpretation of M1 requires a model to determine whether a sufficient number of hazards have been identified.

[6] M1 is an adaptation of M3–Percent Software Safety Requirements, developed by Victor Basili et al. at [36].

$$PSH = (\#Software\ Hazards / \#System\ Hazards) * 100$$

Equation 1. Percent Software Hazards

The model for M1 requires an Estimated PSH (EPSH) based on previously developed similar systems (e.g., similar levels of software control, safety-criticality, and risk levels).

If $|PSH – EPSH| < \sigma$, then a sufficient number of software hazards have been identified in hazard identification, where EPSH is the average of the PSHs for all other similar systems, and σ is the standard deviation of the PSHs.

M2: Software Hazard Analysis Depth. Hazardous software, or safety-critical software, allocated as high- or medium-risk, according to the SHCM, requires analysis of second- and third-order causal factors (should they exist). The depth of hazard analysis, indicated by this metric, will contribute to the sufficiency of overall software hazard analysis and hence validity of derived requirements. M2 will be an indicator of whether hazards have been analyzed to a sufficient depth.

To determine the depth of hazard analysis a Hazard Analysis Space (HAS) must be identified. Complete coverage of the HAS indicates complete hazard causal factor analysis (down to third-order causal factors). Table 1 depicts an example of HAS.

H_1	**H_n**	**Med_1**	**Med_n**	**Mod_1**	**Mod_n**	**L_1**	**L_n**
1	1	1	1	1	1	1	1
2	2	2	2				
3	3	3	3				

Table 1. Hazard Analysis Space

In Table 1 H_n, Med_n, Mod_n, and L_n indicate *n* number of hazards within the same risk category. The numbers reflect levels of analysis. For example, a 3 indicates third-order analysis. As indicated, moderate- and low-level risk hazards do not typically require analysis beyond first-order causal factors. For moderate-risk hazards, other metrics identified in the framework indicate first-order analysis as a matter of course. Low-risk hazards are those which are perceived to be of acceptable nature. Therefore, this metric will only consider high- and medium-risk hazards. When developing the metric it is important to ensure that high- and medium-risk hazards (according to the SHCM) are initially reported separately to allow for subsequent investigation activities.

Calculating the total HAS for each category of hazards (high or medium) is shown in the following formulas.

$$HAS_H = \text{\# High-Risk Software Hazards} * 3$$

Equation 2. Hazard Analysis Space—High-Risk

$$HAS_M = \text{\# Medium Risk Software Hazards} * 3$$

Equation 3. Hazard Analysis Space—Medium Risk

In each equation, the constant 3 reflects the three possible levels of causal factor analysis.

Determining the coverage of hazard analysis (in terms of coverage of the HAS) for each category requires a Hazard Analysis Achieved (HAA) figure expressed as a percentage of total HAS. Equations for high- and medium-risk are given below.

$$C_H = [(\sum HAA_n) / HAS_H] * 100\%$$

Equation 4. Hazard Analysis Space Coverage—High-Risk

$$C_M = [(\sum HAA_n) / HAS_{MED}] * 100\%$$

Equation 5. Hazard Analysis Space Coverage – Medium Risk

In both equations above, C is the *coverage* of the HAS, and HAA_n is the hazard analysis achieved for each risk applicable to their specific categories. HAA_n is determined for each hazard by adding one for each level of analysis. For example, second-order analysis corresponds to a HAA of 2 for a single risk.

The coverage metric identified above provides an indication of depth of hazard analysis for each category of hazards. However, a single overall metric should be provided as an initial indicator of validity of safety requirements. Software Hazard Analysis Depth (SHAD) is determined by averaging the coverage metrics, C_H and C_M.

$$SHAD = (C_H + C_M)/2$$

Equation 6. Software Hazard Analysis Depth

As with other metrics used in this framework, growth or reduction can be best presented through a graph shown over time. However, care must be taken in interpreting the results of this metric. Contrary to other identified metrics, SHAD is not expected to reach 100%. This is because it is highly unlikely that every high- or medium-level risk hazard will have up to third-order causal factors. Hazard analysis may reveal only second-order causal factors in many cases. Another factor that will contribute to reduction of coverage is when new high- or medium-risk hazards are initially identified without immediate hazard causal factor analysis. This will lower the SHAD in the first instance.

Therefore, SHAD is only intended to be an *indicator* (as are all other metrics) to sufficiency of software hazard analysis, and hence an indicator of validity of software safety requirements. Once a baseline is established in the early stages of development, preferably at a point where hazards are well understood, changes in SHAD will require investigation to determine if the depth of analysis *is* sufficient, is *expected* to become sufficient, or *is not* sufficient. SHAD in itself cannot directly determine sufficiency; rather, it is an indicator of sufficiency.

M3: Percent Software Safety Requirements [36]. M3 (PSSR[7]) is an indicator of how sufficient hazard analysis has been performed, and hence the validity of the derived safety requirements. It is similar in format to M1. Interpreting PSSR requires a model that indicates sufficiency, providing upper and lower boundaries. The model for PSSR relies on the previous results of similar systems to determine what is sufficient.

$$\text{PSSR} = (\#\text{Software Safety Requirements} / \#\text{Software Requirements}) * 100$$

Equation 7. Percent Software Safety Requirements

The model for PSSR requires an Estimated PSSR (EPSSR) based on previously developed similar systems. If $|\text{PSSR} - \text{EPSSR}| < \sigma$, then a sufficient number of software safety requirements have resulted from the hazard analysis, where EPSSR is the average of the PSSRs for all other similar systems, and σ is the standard deviation of the PSSRs.

Another possibility for the model is by using a proxy: EPSSR = (#system safety requirements / #system requirements)*100, and σ = X% of EPSSR. X% will require engineering judgment to determine an acceptable deviation from the mean. Success of previous systems, particularly where there has not been a sufficient sample size to determine a standard deviation, should also be taken into account to give an EPSSR. Basili et al. [25] recommend a deviation of 20% EPSSR.

M4: Percent high-risk software hazards with safety requirements. M4 reveals whether any high-risk software hazards have not resulted in applicable safety requirements through hazard analysis. This indicates sufficiency of the process (through artifacts), and hence validity of the requirements.

$$M4 = (\#\ SH_{HR\text{-}SR} / \#\ SH_{HR}) * 100$$

Equation 8. Metric 4

[7] Original PSSR metric uses **e** = σ. For clarity, only σ is used to indicate deviation.

In Equation 8, # $SH_{HR\text{-}SR}$ is the number of high-risk software hazards with associated safety requirements, and # SH_{HR} is the total number of high-risk software hazards.

As development of the system progresses, it is expected that this metric will approach, and reach, 100%. This metric will benefit from being graphed over time.

M5: Percent medium risk software hazards with safety requirements. M5 is simply an extension to M4 by considering medium risk software hazards.

$$M5 = (\# SH_{MR\text{-}SR} / \# SH_{MR}) * 100$$

Equation 9. Metric 5

In Equation 9, # $SH_{MR\text{-}SR}$ is the number of medium risk software hazards with associated safety requirements, and # SH_{MR} is the total number of medium risk software hazards.

M6: Percent moderate risk software hazards with safety requirements. M6 again is an extension to M4 by considering moderate risk software hazards.

$$M6 = (\# SH_{MoR\text{-}SR} / \# SH_{MoR}) * 100$$

Equation 10. Metric 6

In Equation 10, # $SH_{MoR\text{-}SR}$ is the number of moderate risk software hazards with associated safety requirements, and # SH_{MoR} is the total number of moderate risk software hazards.

M7: Percent software safety requirements traceable to hazards. Ensuring traceability to *system hazards* increases the validation case. M7 is simply a percentage indicator of traceability of requirements. All derived software safety requirements must be traceable to system hazards.

$$M7 = (\# SSR_{TR} / \# SSR) * 100$$

Equation 11. Metric 7

In Equation 11, $\# SSR_{TR}$ is the number of traceable software safety requirements, and # SSR is the total number of software safety requirements. This metric should approach, and reach, 100% over time, and will benefit from being graphed.

In total, seven metrics have been identified in the development of the Validation Metrics Framework. Discussed throughout this chapter is the notion of metrics only being an indicator of validity of software safety requirements. This is further discussed in a more concrete example in the following chapter. The key to the Validation Metrics Framework is that the metrics will only act as indicators for further investigation. The investigation itself will determine the actual validity of the software safety requirements, with metrics acting as initiators.

In the following chapter, application of the framework is explored through a case study, with the aim of clarifying the framework and the metrics.

THIS PAGE INTENTIONALLY LEFT BLANK

V. APPLICATION OF THE FRAMEWORK THROUGH CASE STUDY

A. INTRODUCTION

The proposed Validation Metrics Framework developed in Chapter IV is at this stage a theoretical academic exercise. Although it employs some relatively proven and successful methods as its foundation, further analysis is necessary to demonstrate its validity and the benefits to be accrued from its application. This chapter presents a case study of a fictitious surface-to-air missile system as a safety-critical software-intensive system. The system will be based on unrestricted information available in open literature representative of typical surface-to-air missile systems. The case study will work through parts of the system development lifecycle; however, it will not be a complete case study addressing every aspect. Instead it will simply provide a brief look at how the Validation Metrics Framework can be applied throughout the software development process and some of the associated benefits.

B. RASAM[8]

The Rapid Action Surface-to-Air Missile (RASAM) system—a Shipboard Self-Defense Missile System—will be the subject of this case study. The following system description and operational requirements are not intended to be as complete as a “real” system would be. The RASAM will only be defined and analyzed to a sufficient level to allow the Validation Metrics Framework to be exercised. In many cases only representative examples will be called upon, rather than a complete analysis or design.

The RASAM will be followed through a rudimentary tailored waterfall software development lifecycle process. Since this is only a fictional system, we

[8] The RASAM example is taken from the Naval Postgraduate School’s course on Weapon System Software Safety.

are able to employ such a design approach, particularly as we are not fully defining or developing the system. In the sense that requirements for the system shall be known and complete throughout the development process (i.e., there will be no iterative development), the RASAM development will be of waterfall fashion.

Accompanying the development lifecycle will be the system and software safety process as presented by Leveson [14]. It will be the products of this process that will be candidate for measurement by the Validation Metrics Framework.

The aim of this case study is not an in-depth analysis of system design or lifecycle process. Rather, it is simply to show how the framework is applied to a system and how results can be interpreted. Therefore, the complete lifecycle will not be explored.

C. SYSTEM CONTEXT AND ARTIFACTS

The RASAM is initially presented through operational requirements (high-level customer/stakeholder requirements), a series of use cases, misuse cases, and accompanying safety artifacts. The artifacts shown are representative of actual artifacts. They are not intended to be complete or entirely accurate. Metrics used throughout the case study may not correlate with artifacts shown Appendix A. The primary goal of the artifacts is simply to provide context for the system and to represent what should be expected of a real safety-critical system.

1. Operational Requirements

The RASAM system is a Shipboard Self-Defense Missile System. The basic high-level operational requirements for the system are given below. These requirements are equivalent to high-level customer/stakeholder requirements, describing their expectations and desires of the system. Although these requirements will not be utilized in the framework itself, they are given to aid in providing context of the safety-critical software-intensive system to the reader.

The Validation Metrics Framework will only provide indications to validity, identifying critical areas that may require further investigation to ascertain validity of the requirements.

OR.1. RASAM shall be installed on surface combatants, aircraft carriers, and amphibious ships, 8K tons and up.

OR.2. RASAM shall have an effective range of 15km.

OR.3. RASAM shall be an anti-ship missile and anti-aircraft weapon for current and postulated threats.

OR.4. RASAM shall achieve a P_k (probability of kill) not less than .975 for dual salvo.

OR.5. RASAM shall achieve a P_a (availability) not less than .98

OR.6. The Launching System shall:

- Be capable of a minimum of ten 2-missile salvos without reloading,
- Have a shipboard reload capability,
- Be fully reloaded in less than thirty seconds by no more than three trained weapons technicians, and
- Minimize ship alterations required for installation.

OR.7. The Control System shall:

- Interface to existing and future C&C systems and shipboard sensors,
- Provide a stand-alone operation capability, and
- Provide an organic training capability.

OR.8. The launcher subsystem shall have a No-Point/No-Fire design toward the ship's superstructure, equipment, or other places where a missile could impact. It must accommodate moveable equipment:

- The RASAM shall have automatic detection of equipment movement.

It must also accommodate ancillary equipment such as aircraft and helicopters:

- The RASAM shall preclude firing in the direction when deck areas are occupied.

2. System Use Cases

The system use cases further describe how the customer/stakeholders intend to interact with the RASAM and the expected operations and results. Each use case is described from the users' perspective at the early stages of development, and is therefore not "fully-dressed." These use cases will provide sufficient depth for the purpose of this case study. The use cases can be found in Appendix A.

3. System Misuse Cases

The system misuse cases describe how the system shall react to situations whereby the system is used incorrectly. In terms of safety, a system misuse case will typically focus on incorrect unintentional operation rather than malicious operation. However, depending on the type of system, malicious system misuse cases concerning safety will also be generated. Like the system use cases, they are not "fully-dressed," but are sufficient for understanding the context in the case study. The system misuse cases can be found in Appendix A following the system use cases.

4. System Description

Based on the high-level user requirements, use cases and other user documentation, a basic description of the RASAM system can be found at Appendix A. The description covers the system components that enable required functionality. The system description is essentially the product of the conceptual design phase. Typically, this would be accommodated by system requirements, or simply described through these requirements. However, for the purpose of this thesis, a simple description will suffice.

5. Safety Artifacts

During the conceptual/preliminary design, a Preliminary Hazard List (PHL), Preliminary Hazard Analysis (PHA), generic safety requirements, derived safety requirements, and System Software Safety Requirements Traceability Matrix (SSSRTM) are each developed. These safety products can also be found in Appendix A. Each safety product would be expanded upon in subsequent design phases and, although they will technically not be "preliminary" following preliminary design, they will retain their nomenclature throughout the case study. These artifacts will be the subject of measurement throughout the system development, and, where necessary, will be updated throughout the development process.

D. VALIDATION METRICS FRAMEWORK APPLICATION

The application of the Validation Metrics Framework will vary depending on different factors influencing the goal structure. The safety team may enhance or change the core goal structure as presented in this thesis to suit the team's particular application—it is extensible by nature. However, by at least using the framework as-is, a much more complete coverage of validation, aided by metrics, will be achieved than with current validation methods. During the conceptual design phase, most safety artifacts will focus on hazard identification. Some hazard analysis can be performed, and safety requirements derived, but lack of a full system design will prevent complete safety design. As the system matures throughout the lifecycle, a better understanding of system functionality, and subsequent safety analysis, is obtained.

The safety product reference list consists of:

1. RASAM Preliminary Hazard List.

2. RASAM Preliminary Hazard Analysis.
3. RASAM Generic Safety Requirements.
4. RASAM Derived Safety Requirements.
5. RASAM SSSRTM.

In accordance with the Validation Metrics Framework goal hierarchy shown in Figure 14, the following framework inputs can be identified for the RASAM system:

- SDLP: Waterfall—requirements stability means that understanding of system functionality, and hence safety measures, will be obtained early in the development lifecycle.
- Safety Inputs: As per safety product reference list above.
- Stakeholder Feedback: Minimal due to SDLP; however, will still be an extremely important aspect of validation. Stakeholder feedback should be requested whenever metrics lead to investigations of validity.

To reiterate, the overarching Framework Goal is:

- **Purpose:** To measure the products of the software safety process throughout development and implementation in order to aid in validating software safety requirements.
- **Perspective:** Examine the metrics from the safety engineering team's point of view, with a focus on validating software safety requirements by proxy—in accordance with the proposed validation of safety requirements model.
- **Environment:** The system has safety-critical elements that will be bound by the software safety process.

1. Metric 1: Percent Software Hazards

Utilizing the framework metrics identified in the previous chapter, we begin with Percent Software Hazards—ensuring that a sufficient number of hazards have been identified. The following table provides historical metric data from previously developed similar systems.[9] It includes the number of hazards identified for each and the number of software hazards derived. It is assumed that only "successful" systems (i.e., those perceived to be successful in the eyes of both the developer and customer) will be used for determining the model for M1. It must also be noted that the data presented here is obtained from the finalized system, not from early development phases.

System	# System Hazards	# Software Hazards	PSH
RASAM v0.5	145	70	48.3%
ISAM (Integrated SAM)	110	50	45.5%
SAMDS (SAM Defense System)	95	34	35.8%
AAMS[10] (Air-Air Missile System)	165	105	63.6%
RAAMS (Rapid AAMS)	198	104	52.5%

Table 2. Case Study PSH

Based on the PSH for each system given in Table 2, the mean and standard deviation can be calculated. The standard deviation is given by using the root-mean-square (RMS) method assuming a complete population rather than a "sample" population. This results in a smaller deviation and therefore, according to previous definitions, a more sufficient number of identified hazards

[9] The data presented for the "similar" systems is representative of real systems, however, is fictitious.

[10] For the purpose of this thesis it is assumed that the Air-Air Missile System and the Rapid AAMS both have a similar level of software control, similar software safety development process, and similar hazard design space.

due to tighter boundaries.[11] Also, it can be assumed that, where data is available, the entire population of available *similar* safety-critical software-intensive systems would be used, rather than just a sample of the data.

Metric 1 (M1) data is as follows:

- **EPSH = 49.1%**
- **σ = 9.1%.**

At the end of the system design phase (final metric sample) the hazard identification process resulted in **180 system hazards** being identified and recorded in the preliminary hazard identification document. Out of this 180 total hazards, **68** are directly associated with software control. Arriving at the number of 68 requires that Preliminary Hazard Analysis (PHA) be performed, identifying which system hazards are directly influenced by software control. This figure does not look at lower-level software causal-factor hazards; rather, it is intended to ensure that a sufficient number of hazards are associated to software based on levels of software control and safety-criticality of similar systems.

PSH = (68/180) * 100 = 37.8%.

Using the data above, it can be shown that the RASAM PSH is outside of one standard deviation from the EPSH:

|PSH – EPSH| = |37.8 – 49.1| = 11.3% > σ.

Throughout the system development, the following data, summarized in Table 3, was collected regarding the RASAM PSH.

[11] The standard deviation is smaller, thus reducing the error allowed within the model for sufficiency.

M1 Sample[12]	**# System Hazards**	**# Software Hazards**	**PSH**
1	140	20	14.3%
2	164	45	27.4%
3	171	67	39.2%
4	180	68	37.8%

Table 3. RASAM PSH Samples

Tracking M1 is best achieved by graphing each sample to monitor the growth, or reduction, of PSH.

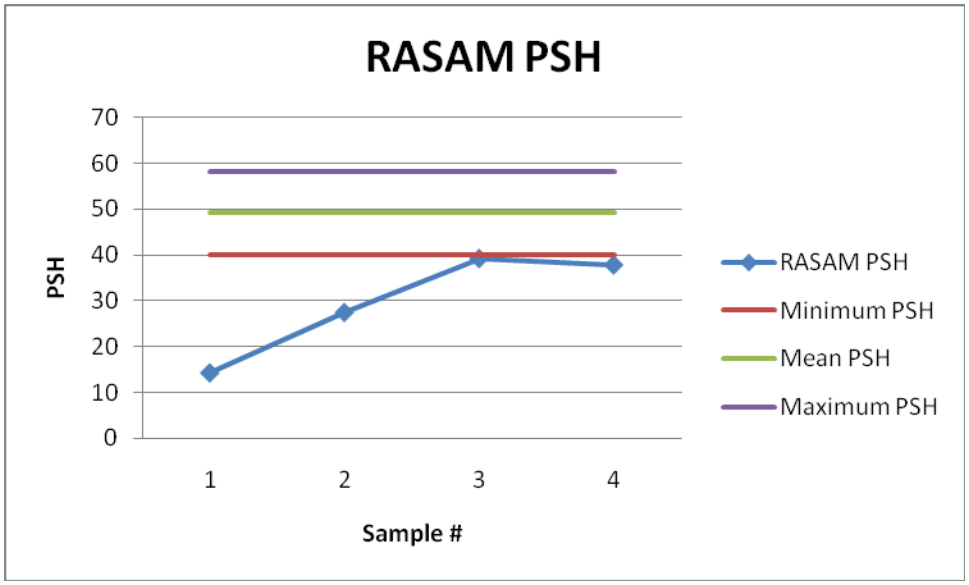


Figure 21. RASAM PSH Growth

Figure 21 indicates that the PSH has grown[13] significantly as the system has progressed through the development lifecycle. This type of growth is

[12] The sample numbers represent samples taken at various stages of development as time progresses, i.e., sample 1 is taken before sample 2, etc. These sample numbers, and assumed positions in time, remain constant throughout the case study.

[13] Any reference to "growth" or "growth model" in this thesis refers to growth or reduction based directly on observed data. It does not consider predictive growth models.

expected as initially hazard identification will not identify which hazards are associated with software. After PHA we can expect the PSH to grow. Often the conceptual phase will not result in mature system data as the design is not well understood. This could be the reason for an initially insufficient number of software hazards being identified. However, growth from samples 1 through 3 indicates that PSH is “on track.” Sample 4 shows a reduction in PSH, which is not expected. As it stands, the results of M1 indicate that further investigation needs to be made to determine the sufficiency of hazard identification, and hence the validity of the resulting requirements. The result may not be a cause for concern as it would be expected that the system would be better understood as it continues through the development lifecycle. Therefore the PSH may increase to a sufficient level. Regardless, an investigation must be conducted to determine the cause of the reduction and plan for the necessary action to guarantee growth into the sufficiency band (i.e., between minimum and maximum EPSH).

2. Metric 2: Software Hazard Analysis Depth

The data given in Table 3 provides a starting point for SHAD. This data indicates the total number of software hazards (causal factors) from which high- and medium- risk hazards must be obtained. Table 4 provides representative data for the first portion of calculating the SHAD.

Hazard	Analysis Depth	Hazard Analysis Space (HAS)
H1 (High)	3	HAS_H = # High-Risk Software Hazards * 3 **HAS_H = 6 * 3 = 18**
H2	2	
H3	3	
H4	3	
H5	2	
H6	1	
M1 (Medium)	2	HAS_M = # Medium Risk Software Hazards * 3 **HAS_M = 5 * 3 = 15**
M2	3	
M3	2	
M4	2	
M5	3	

Table 4. SHAD Data

Utilizing the Hazard Analysis Achieved (HAA) data in the above table, the sum of HAA is shown below:

High-Risk: **$\sum HAA = 3 + 2 + 3 + 3 + 2 + 1 = 14$.**
Medium Risk: **$\sum HAA = 2 + 3 + 2 + 2 + 3 = 12$.**

Given HAA for each category, C_H and C_M can now be calculated:

$C_H = [(\sum HAA_n) / HAS_H] * 100\% = [14/18] * 100\% = 78\%$.

$C_M = [(\sum HAA_n) / HAS_{MED}] = [12/15] * 100\% = 80\%$.

SHAD can then be determined as the average of the two coverage metrics to give an overall coverage of the entire HAS (without a biased weighting to the high-risk category as it has a larger number of hazards):

SHAD = $(C_H + C_M)/2 = (78\% + 80\%) / 2 = 79\%$.

The first instance of SHAD is found to be 79%. As stated previously, 100% is an unrealistic figure (although possible). Therefore, 79% may prove to be a sufficient baseline. However, to prove sufficiency of this baseline figure, the safety team must ensure that the depth of analysis on each hazard has been performed adequately through investigation.

Table 5 provides sample data for SHAD throughout the development of the RASAM system.

M2 Sample	# High-Risk	# Med Risk	C_H	C_M	SHAD
1	6	5	78%	80%	79%
2	10	12	73%	92%	82.5%
3	15	20	67%	83%	75%
4	18	21	89%	87%	88%

Table 5. SHAD Data Samples

A combined graph of C_H, C_M and SHAD is shown in Figure 22 detailing the fluctuation in hazard analysis depth.

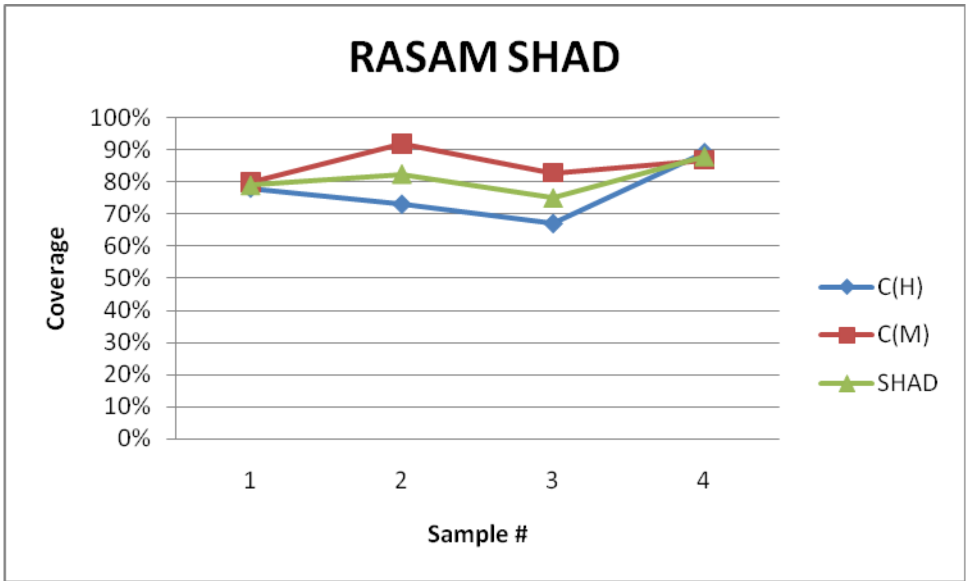


Figure 22. RASAM SHAD Growth

The above figure shows a distinct decline in SHAD at sample 3. At this same sample point there is a decline in both C_H and C_M. However, investigation into the cause of this decline would require determining the cause. From the graph, it can be assumed that correction took place prior to sample 4, resulting in an increase in SHAD. As discussed earlier, the SHAD metric will only provide indication of sufficiency for further investigation.

3. Metric 3: Percent Software Safety Requirements

Utilizing the same set of *similar* safety-critical software-intensive systems found at Metric 1, Table 6 shows data for the PSSR model.

System	# Software Requirements	# Software Safety Requirements	PSSR
RASAM v0.5	94	25	26.6%
ISAM (Integrated SAM)	78	31	39.7%
SAMDS (SAM Defense System)	109	43	39.4%
AAMS (Air-Air Missile System)	120	56	46.7%
RAAMS (Rapid AAMS)	132	51	38.6%

Table 6. Case Study PSSR

Based on the same assumptions made for M1 (complete population vs. sample of population), PSSR (M3) data is as follows:

EPSSR = 38.2%.

σ = 6.5%.

At the end of the system design phase, the hazard identification and hazard analysis resulted in a total of **44 software safety requirements** being derived. The total software requirements completed at the same time was **105**. PSSR can be calculated as follows:

PSSR = (44/105) * 100 = 41.9%.

Analyzing the PSSR through the sufficiency model, it can be shown that, for the current number of software requirements, there are a sufficient number of software safety requirements:

|PSSR – EPSSR| = |41.9 – 38.2| = 3.7% < σ.

From the above data it can be assumed with reasonable confidence that a sufficient number of software safety requirements have been derived based on the current number of software requirements. Thus, this strengthens the validation case for the derived software safety requirements.

In the same fashion as M1, a number of samples were taken throughout the conceptual design phase as shown in Table 7.

M4 Sample	# Software Requirements	# Software Safety Requirements	PSSR
1	83	15	18%
2	94	37	39.4%
3	101	40	39.6%
4	105	44	41.9%

Table 7. RASAM PSSR – Conceptual Phase

As with M1, tracking the growth or reduction of M3 is best achieved by graphing the results over time. This will aid in indicating maturing (or otherwise) valid software safety requirements.

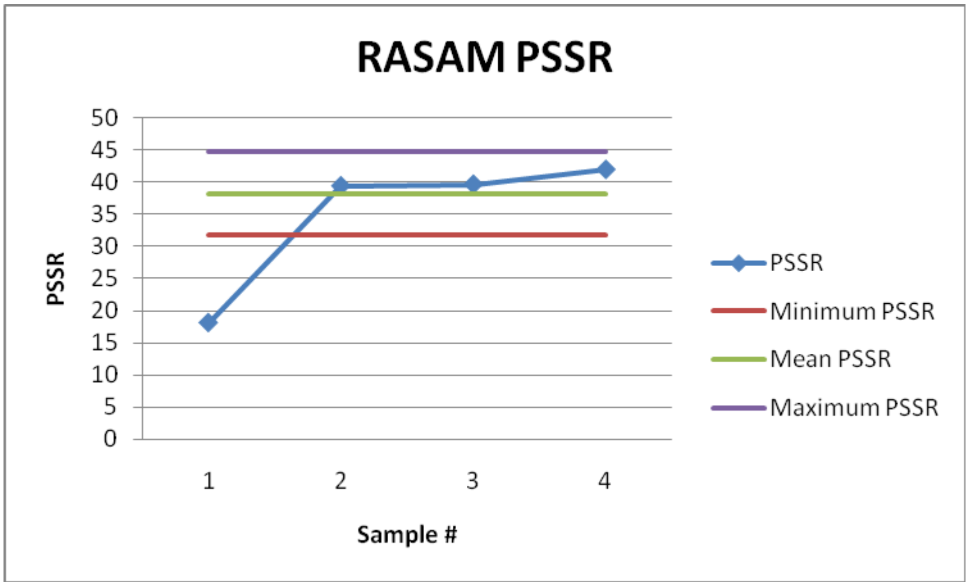


Figure 23. RASAM PSSR Growth

Figure 23 and the calculated PSSR, indicate a sufficient number of software safety requirements are being developed. Thus, instilling confidence that the safety requirements are indeed valid.

4. Metric 4: Percent High-Risk Software Hazards with Safety Requirements

It is assumed that M4 will be 100% throughout the development cycle; that is, every identified high-risk software hazard will be mitigated through appropriate software safety requirements. However, it is possible that certain elements of the design are forgotten, or postponed for later development. Therefore, it is important to ensure that every high-risk software hazard is associated with derived software safety requirements. The following table is the metric data used for M4, M5, and M6.

Sample #	# SH_{HR-SR}	# SH_{HR}	# SH_{MR-SR}	# SH_{MR}	# SH_{MoR-SR}	# SH_{MoR}
1	2	4	5	14	7	8
2	6	9	12	18	14	14
3	10	11	23	24	19	20
4	12	12	24	24	20	22

Table 8. Software Hazards with Safety Requirements Data

Using the data given in Table 8, M4 can be calculated for the final sample as:

M4 = (# SH_{HR-SR} / # SH_{HR})*100 = (12/12)*100 = 100%.

This result indicates that, at the end of the system design, all high-risk software hazards are associated with software safety requirements, indicating partial validity of the derived requirements.

5. Metrics 5 & 6: Percent Medium Risk Software Hazards with Safety Requirements, and Percent Moderate Risk Software Hazards with Safety Requirements

Following on from M4, M5 and M6 are simply extensions. Therefore, it will be more beneficial, and efficient, to combine all three metrics (M4, M5, and M6) into a single graphic. Using the metric data in Table 8, the following table shows the results of M4, M5, and M6 throughout development.

Sample #	M4	M5	M6
1	50%	36%	88%
2	67%	67%	100%
3	91%	96%	95%
4	100%	100%	91%

Table 9. Percent Software Hazards with Safety Requirements

Graphing the sample sets gives a more clear depiction of growth over time and indicates in one simple diagram which area (high-, medium-, or moderate-risk) of software safety requirements are sufficiently mitigated, and which areas of identified software hazards may be invalid. Overall, the metrics give an insight into the sufficiency of the artifacts resulting from the safety process, and hence the validity of the requirements.

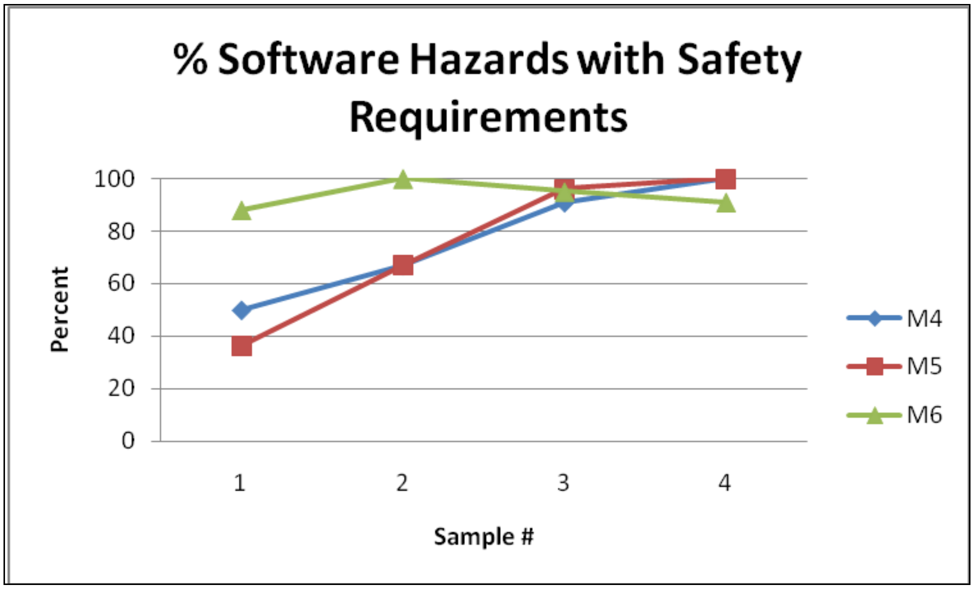


Figure 24. M4, M5, and M6 Growth

Figure 24 above indicates that all high- and medium-risk software hazards have associated software safety requirements. However, there are some moderate-risk software hazards that are not mitigated through software safety requirements. Therefore, investigation is required to determine either the validity of the software hazard, or the validity of the set of software safety requirements.

6. Metric 7: Percent Software Safety Requirements Traceable to Hazards

As with the previous three metrics (M4, M5 and M6), it is expected that all software requirements are traceable to hazards. In essence, the combination of M4, M5, M6, and M7 ensures forward and backward traceability. It may be the case that all software hazards have associated software safety requirements;

however, there is no guarantee that the complete set of derived software safety requirements is traceable to software hazards. At the end of system design it was found that, as with M3, the total number of software safety requirements was 44. Each of these software safety requirements was traceable to identified software hazards. Metric 7 calculations are as follows:

$M7 = (\# SSR_{TR} / \# SSR)*100 = (44/44)*100 = 100\%$.

Table 10 provides the metric data for M7 with the same four collection sample points as previous metrics.

Sample #	**# SSR_{TR}**	**# SSR**	**M7**
1	15	15	100%
2	36	37	97.3%
3	38	40	95%
4	44	44	100%

Table 10. Traceable Software Safety Requirements Data

Again, graphing over time gives us an indication of stability, or growth, or otherwise. In the case of M7, stability at 100% is the goal. Anything other than this would indicate the need for investigation.

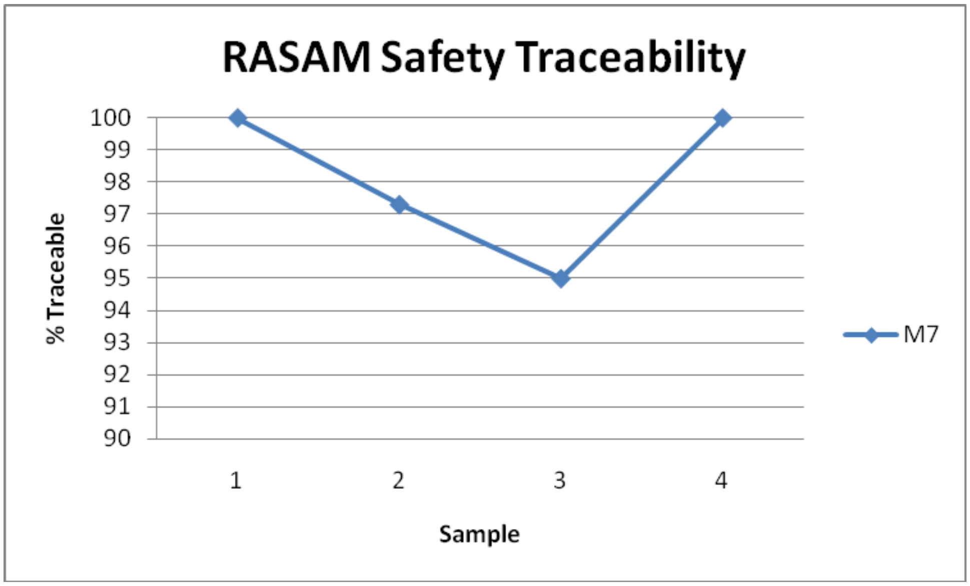


Figure 25. Percent Software Safety Requirements Traceability

From the above metric data it is shown that, at the end of the system design, all software safety requirements are traceable to software hazards. This, again, strengthens the case that the derived software safety requirements are valid.

E. CASE STUDY CONCLUSION

In the above sections, the application of the Validation Metrics Framework has been demonstrated. As has been discussed, the metrics themselves cannot determine validity of the resultant safety artifacts (ultimately focusing on software safety requirements), but they do provide an indication of validity in a proactive manner. Instead of relying on final testing to reveal any validity issues with software safety requirements, application of the framework helps to identify potential problems early on in the development lifecycle.

Upon finalization of system design in the case study, some metrics (M1, M2, and M6) indicate that further investigation must be made to determine the source of insufficiency, and the possibility of invalid software safety requirements. Throughout the development process, up to the final sample set, a number of investigations would have taken place to remedy, or understand, metrics indicating poor performance of the assessed entity. Stakeholder feedback is necessary during these investigations, providing a closed-loop process for determining validity of software safety requirements. The case study only presents four sample sets of measurement data up to finalization of the system design. There is no “standard” number of samples that should be considered for system development; rather, it should be determined according to a number of factors, including but not limited to the following:

- System complexity
- Workforce
- Metric performance (poor performance indicates the need for close monitoring and more regular samples)
- Metric gathering burden
- Progress through SDLP (up front effort usually outweighs trailing effort).

VI. CONCLUSION

A. KEY FINDINGS AND ACCOMPLISHMENTS

Validation as a concept is often misunderstood. Too often it is confused with verification activities, or left as a final box to tick just prior to delivery. Adding to this is the difficulty of validating software safety requirements. Validation of software safety requirements cannot be performed in the usual sense, which relies heavily on stakeholder input, because of the disconnect that exists between stakeholders specifying that they want a safe system and stakeholders understanding *how* to make a system safe. Ensuring that the right safety product is built is vital to the successful deployment and acceptance of a software system. When considering the possible impacts of a safety-critical software-intensive system, validation of safety requirements is paramount.

Validation of software safety requirements necessitates a new model of validation. Chapter III of this thesis proposed a new model for validation of safety requirements, focussing on sufficiency of hazard identification, hazard analysis, and software safety requirements traceability as a proxy for validation. This model forms the core of the proposed Validation Metrics Framework.

At present very little information exists on the use of metrics for the purpose of measuring safety. Even less, if any, information can be obtained on metrics for validation. Discussion on what validation metrics are, and how best to use them, is given throughout this thesis. By combining two popular software development tools (GQM and GSN) we have created a goal-based framework identifying a core set of metrics to aid in validating safety requirements of safety-critical software-intensive systems.

The framework cannot be claimed to be complete. It is an initial step into a more mature software engineering environment utilizing measurement techniques. Rather than completeness, the framework should be considered in terms of sufficiency. Sufficiency of the framework will depend on the application and surrounding organizational environment. Each organization will determine different levels of sufficiency based on a number of contributing factors, such as:

- Experience with validation of software safety requirements
- Experience with the framework itself
- Organizational acceptance of a metrics framework
- Overhead burden of gathering metrics
- Complexity of the system under design
- Relative success of previous designs.

The Validation Metrics Framework has been designed in an extensible manner. Addition of new goals, questions, or metrics is not limited. Should better methods of measuring sufficiency of any one of the three elements of validation (i.e., hazard identification, hazard analysis, and software safety requirements traceability) be determined, the framework will allow for modification. Tailoring to suit any organization's requirements is possible, but keep in mind that it is expected that in most cases this would involve an addition to the set of metrics. The metric set presented in the Validation Metrics Framework is given as a core set—they have been developed to be the minimum measures of sufficiency. However, this does not prevent partial application of the framework where benefits can be obtained.

The research and development of this thesis resulted in a metric framework for validation of safety-critical software-intensive systems. There is currently no notion of validation metrics in the open literature, much less a framework identifying purpose, application, and boundaries of the metric set.

Application of the framework to a representative safety-critical software-intensive system (i.e., the RASAM) in Chapter V shows how the resulting metric data from the framework can engender a proactive approach to ensuring validity of software safety requirements. The Validation Metrics Framework identifies a number of areas that require investigation throughout the case study. Moreover the results of the case study demonstrated that the framework provides early warnings of the invalidity of software safety requirements. As noted throughout the case study, the metrics cannot determine validity of the safety requirements themselves. Instead they serve as indicators (in some cases early indicators) of software safety requirements validity. Only subsequent investigation, triggered by the metrics in the framework, can determine if the software safety requirements are in fact valid.

B. FUTURE WORK

The primary avenue of future work is to apply the Validation Metrics Framework to a number of real systems under development. Although a metric for effectiveness of applying the framework has not been identified, a survey could be used to determine effectiveness as perceived by both developers and stakeholders. Establishing effectiveness via perception would provide justification for application and provide a firm grounding for continued use and development of the Validation Metrics Framework.

As mentioned in the previous paragraph, a metric to measure the effectiveness of the framework has not been identified. Initially a qualitative metric should be developed with an aim to move to an automated quantitative metric. Effectively, a Return On Investment (ROI) style quantitative metric is required to provide even further justification for the framework. The ROI metric may only consider resources as measures of success. Thus, a qualitative metric is still required to determine effectiveness in terms of validity (as per the previous paragraph). Therefore, a two-part metric is required to measure effectiveness of the framework—a quantitative ROI portion (concerned with resources) and a qualitative portion rating the perceived effectiveness.

At present, the framework determines validity of safety requirements through sufficiency measures. Although carefully researched, it is expected that more appropriate and effective sufficiency measures may be identified that still maintain the core goal-set as identified through this thesis. Application of the framework to real systems will most likely result in improvements to the current core metric set, or identify new metrics. There are likely to be many other ways to measure sufficiency than those proposed in this thesis.

The current Validation Metrics Framework specifically addresses safety-critical software-intensive systems. Many safety-critical system developers are now moving toward system-of-systems solutions. The aim of these solutions is to leverage off previous development (in many cases) while creating an overall system that has capability beyond the “sum of its parts” (i.e., synergy of systems). Designing safety into a system-of-systems can be a much more complex task than that of standard systems. Until recently, hazard identification and hazard analysis methods were unable to deal with system-of-systems (see [26] for a detailed analysis of the system-of-systems problem when identifying and analyzing safety hazards). Redmond proposes an interface hazard analysis technique for systems-of-systems and identifies a number of other hazard identification and analysis areas as future work. The Validation Metrics Framework is highly reliant on the methods of identifying and analyzing hazards. Therefore, a complete hazard analysis method is required for systems-of-systems before applying the framework to systems-of-systems. Because these types of systems are only recent advances in software engineering, success of design has been varied. For many of the metrics identified in the Validation Metrics Framework, historic data is required from successful systems to ensure valid metric interpretation. Presently, this historical data is not readily available. Therefore, when dealing with system-of-systems solutions, two hurdles must be cleared:

- Maturing of hazard identification and analysis
- Compilation of historical metric data.

Once the above objectives are met, there is no foreseeable reason why the Validation Metrics Framework cannot be applied with the current metric set. This application will require further investigation to define clear-cut processes and definitions (e.g., there may be two different system specifications resulting in a complex web of requirements traceability, but still needs to be maintained).

The Validation Metrics Framework is specific for safety-critical software-intensive systems. Through the development of this framework, a hybrid generic metrics model has also been identified through the combination of GQM and GSN. This hybrid model provides all the advantages of a goal-based metric framework, while extending the GQM structure to include more descriptive elements from GSN. The Validation Metrics Framework for safety-critical software-intensive systems is one example of a validation metrics framework. Future derivatives of the proposed framework could result in validation metrics frameworks for nearly any requirement type or validation aspect. The basic hybrid model is extensible and therefore could also be used for metric frameworks even beyond the realms of validation, for instance verification.

Discussed during the objectives section for the Validation Metrics Framework is the concept of providing a goal ranking system. Because resources are not always plentiful, a realistic view of goal ranking (and subsequent metric ranking) is required. A number of possible solutions exist for goal ranking; however, the method of Berander and Jönsson [19] seems promising. Allowing for surveyed prioritization could be an initial implementation, while working toward a more complete solution that takes into account severity of risks and other factors. In fact, any ranking surveys undertaken need to clearly explain the consequences of rating a certain goal too low. Neglect on any metric that deals directly with high-risk software could result in severely invalid software safety requirements. Future development and research on the Validation Metrics Framework needs to consider prioritization as an integral part of the framework, otherwise the risk of rejecting the framework due to lack of resources is heightened.

Software reuse is another hotly debated topic in the software engineering community. Although it has not been discussed in depth, the reality is that software reuse can deliver benefits to developers of software-intensive systems. It can also result in severe complications during design and failures after delivery. However, given that there is much to be gained from software reuse, it makes sense that any metric framework for modern software-intensive systems should consider its impact. Future development of the Validation Metrics Framework should include analysis of the impact that software reuse can potentially have. Given that the framework is used for validation of software safety requirements, the impact on requirements due to software reuse will be the focus. As part of this analysis, a sufficient method of managing software reuse should be recommended to bound the possibilities of design artifacts and procedures. Warren presents a suitable framework for software reuse in safety-critical system of systems called C^5RA [27].

Automation of the Validation Metrics Framework is also an important aspect. To reduce the burden on metric gathering, tools should be developed to gather, monitor, and regularly report on the framework metrics. To a great extent, automation of the framework can be achieved. However, the intellectual activities—tailoring the framework and conducting investigations into validity of requirements—cannot be automated.

APPENDIX A. RASAM DESIGN ARTIFACTS

A. INTRODUCTION

The documentation compiled in this section is only intended to be representative of an actual system. It is not a complete representation. Sufficient data is provided so that the context is clear; however, when the metrics are presented for the RASAM, the results will not correlate directly with the information below. For example, there are only seven safety requirements identified and fourteen hazards. Not every hazard will be addressed through the shown requirements, though the metrics shown will assume representative data. The purpose is to show how the framework is used, not to fully define a software-intensive system.

Format and content of the following design artifacts will vary depending on the application and/or originator. However, the purpose and context of the documentation will be similar to that given below.

Obtaining metric data, where possible, should be an automated process. This will reduce the overhead burden on any metric program. In most cases, some form of CM tool would be used to obtain this information, and therefore would be in a much more organized fashion than the form below.

B. SYSTEM USE CASES

1. UC1 – Command & Control (C&C) Interfaced RASAM Launch

Scope: RASAM System.

Level: User-Goal.

Primary Actor: C&C Operator (referred to as “User” in Main Success Scenario).

Stakeholders and Interests:

- C&C Operator – Wants successful launch and kill of target.
- Weapon crew – Does not want to engage with RASAM unless ammunition depleted.

Preconditions:

- RASAM interfaced with C&C system.
- Built In Tests (BITs) passed.
- RASAM in tactical mode.
- C&C recognized air/surface picture valid.
- RASAM loader contains sufficient munitions.

Success Guarantee: RASAM launches without incident and achieves P_k of > 0.975 for dual salvo against designated threat within designed capabilities.

Main Success Scenario:

1. User identifies enemy target on C&C system.
2. User initiates RASAM launch through C&C system on identified enemy target.
3. RASAM receives launch command and initiates launch.
4. RASAM deploys dual-salvo attack.
5. C&C system continuously updates target position to RASAM system.
6. RASAM system relays target position to launched munitions.
7. Munitions match position and lock to detected vessels when within range of detection.
8. Munitions impact and destroy target.

Extensions (Alternates):

*. User may, at any stage throughout Main Success Scenario, abort attack:

   a. User issues abort command through C&C system.

b. RASAM receives abort command and initiates abort procedure.

c. If safe to do so, munitions are destroyed in flight.

d. If flight path does not allow for safe self-destruct, RASAM issues new coordinates and enables self-destruct of munitions.

6. Should communication to munitions fail, missile will disarm, de-energize propulsion, and ground.

7. Should position mismatch occur (i.e., relayed position not match detected vessels) from detection range to 1km from detected vessels or C&C target (whichever is closer), error shall be sent to RASAM and missile shall disarm, de-energize propulsion, and ground.

8. Should missiles fail to impact target (only if neither missile impacts):

a. Missile shall attempt to re-track and engage target until onboard fuel < 5% but not < 4%.

b. Missile shall self destruct if fuel load between 4% and 5% if no impact and if safe to do.

c. If not safe to self-destruct, RASAM issues new coordinates and enables self-destruct of munitions.

8. Should one missile impact and not the other, non-impacting missile shall detonate within 50m of other missiles impact.

2. UC2 – Launcher Reload

Scope: RASAM System.

Level: User-Goal.

Primary Actor: Weapon Crew.

Stakeholders and Interests:

- C&C Operator – Wants fast reload of munitions, initiates reload request.
- Weapon crew – Wants easy and simple reload procedure to allow for coordinated reload.

Preconditions:

- RASAM interfaced with C&C system.
- Munitions depleted and signaled to C&C Operator.
- RASAM in standby mode.
- Spare launcher sub-assembly positioned for reload.
- Spare launcher sub-assembly BITs passed.

Success Guarantee: Weapon crew (consisting of at least three members) is able to reload RASAM (twenty missiles) in less than thirty seconds and signal for return to required mode.

Main Success Scenario:

1. C&C Operator requests munitions reload.
2. Weapon crew receives reload request.
3. Weapon crew checks RASAM in standby mode and initiate reload.
4. RASAM lowers depleted launcher sub-assembly.
5. Weapon crew extracts depleted launcher and insert new launcher.
6. Weapon crew activates reload insert.
7. RASAM loads new launcher and initiates reload BITs.
8. RASAM signals C&C operator that reload BITs passed.

Extensions (Alternates):

8. Should the reload BITs not pass, the RASAM should signal the C&C operator and prevent from entering tactical mode.

*. The RASAM shall not change from standby mode throughout the whole reload procedure, and should not change mode until explicitly called upon completion of the procedure.

C. SYSTEM MISUSE CASES

The following misuse case details a sequence of actions that could lead to incorrect target designation. This example of misuse case would be created in the conceptual design of the system, identifying many of the potential system hazards that need to occur to allow such a safety incident. Misuse cases are the primary identifier of system hazards (then recorded in PHL), and also a form of Preliminary Hazard Analysis. However, they are intended to be rudimentary starting points, not detailed analysis tools.

1. MUC1—Incorrect Target Designation

Scope: RASAM System.

Level: User-Goal.

Primary Actor: C&C Operator (referred to as "User" in Main Success Scenario), C&C System.

Stakeholders and Disinterests:

- C&C Operator – Does not identify incorrect designation.
- C&C System – Identifies incorrect target.

Preconditions:

- RASAM interfaced with C&C system.
- BITs passed.
- RASAM in tactical mode.
- C&C recognized air/surface picture valid.
- RASAM loader contains sufficient munitions.

Failure Guarantee: RASAM identifies target according to C&C identification without rejection of incorrect designation.

Main Success Scenario:

1. User incorrectly identifies friendly as enemy target on C&C system.
2. C&C system allows incorrect designation:

a. C&C does not retain Identification Friend or Foe (IFF) coding for target, or

b. If C&C does not have IFF coding, does not interrogate for update.

3. User initiates RASAM launch through C&C system on incorrectly identified enemy target.

4. RASAM receives launch command and initiates launch.

5. RASAM does not perform any IFF interrogation on target throughout launch procedure.

Extensions (Alternates):

*. User may, at any stage throughout main success scenario, abort attack:

a. User issues abort command through C&C system.

b. RASAM receives abort command and initiates abort procedure.

*a. User may, at any stage throughout main success scenario, re-identify the target:

a. User issues re-identification.

b. C&C passes to RASAM.

c. RASAM receives update and re-identifies, whilst maintaining history of target.

D. SYSTEM DESCRIPTION

The following system description serves as background information to provide further context to the RASAM example.

The RASAM design consists of three major subsystems:

- Launcher Subsystem,
- RASAM Missile, and
- Weapon Control System (WCS).

The RASAM also consists of three major ancillary equipment subsystems:

- Loader,
- Launcher Interface Test Kit, and
- Missile Interface Test Kit.

1. WCS

The WCS interfaces to the host command and control system via optical fiber:

- The C&C type can vary according to ship class.
- Accepts target designations and engagement orders from C&C.
- Issues commands to Launcher Subsystem to prepare and launch missile

The major subsystems of the WCS are:

- Interface Adapter Panel (IAP)
    - o Interfaces to host combat system
    - o Interfaces to shipboard sensors
    - o Formats message traffic for SCU.
- Weapon Control Panel (WCP)
    - o Operator interface
    - o Select operational mode (off, standby, test, training, tactical)
    - o Weapon Control for stand-alone operations.
- Launcher Interface Unit (LIU)
    - o Interfaces WCS to Launcher.
- System Control Unit (SCU)
    - o Monitors and controls all system functions
    - o Provides feedback to the operator on system status.

2. Launcher

The launcher subsystem has ready-service stowage for twenty missiles:

- Restrain missiles in launcher against ship's motion up to sea-state 6
- Prepare, point, and fire missiles in direction of threat
- Protect missiles from effects of shipboard environment.

The major subsystems of the Launcher subsystem are:

- Launcher
  - o Stores and protects missiles
  - o Positions missiles for firing.
- Launcher Control System (LCS)
  - o Controls train and elevation of launcher.
- Missile Interface Assembly (MIA)
  - o Interface between WCS and missiles.

3. RASAM Missile

The RASAM Missile is not a unique component, therefore can be considered typical of a shipboard Surface-to-Air Missile.

E. SAFETY PRODUCT REFERENCE LIST

Date	Name	Version
10Apr2004	RASAM Preliminary Hazard List	V1
15Jul2004	RASAM Preliminary Hazard List	V2
15Jul2004	RASAM Preliminary Hazard Analysis	V1
12Oct2004	RASAM Preliminary Hazard Analysis	V2
30Nov2004	RASAM Generic Safety Requirements	V1
10Jan2005	RASAM Derived Safety Requirements	V1
15Feb2005	RASAM Software Safety Requirements Traceability Matrix	V1

Table 11. RASAM Safety Product Reference List

F. RASAM PHL

Identifier	Date	Hazard	Comments
H1	10Apr2004	Detonation or deflagration of warhead	Becomes mishap if unintentional loss occurs
H2	10Apr2004	Explosion or deflagration of rocket motor	
H3	10Apr2004	Loss of control of launched missile	
H4	10Apr2004	Loss of control of launcher movement	
H5	10Apr2004	Exposed high voltages on launcher	

Identifier	Date	Hazard	Comments
H6	10Apr2004	High pressure liquid / gas escape on launcher	Hydraulics / Pneumatics
H7	10Apr2004	Exposed toxic materials	
H8	10Apr2004	Launch debris	Ejected components
H9	20May2004	Unintentional launch	
H10	20May2004	RASAM / C&C position discrepancy	
H11	15Jun2004	Loss of communication with launched missile	
H12	23Jun2004	Loss of communication between C&C and RASAM	
H13	15Jul2004	RASAM launcher reposition for reload	Maintenance personnel around lowered launcher
H14	15Jul2004	Incorrect target designation	Highly dependent on C&C system, but not entirely

Table 12. RASAM Preliminary Hazard List

G. RASAM PHA

For the purpose of this thesis, the PHA will only be presented in terms of mishap risk associated with each of the hazards. In a full PHA, a much deeper analysis of each potential mishap would be performed—including mishap risk, determination of causal factors (in themselves hazards), and suggested mitigation techniques—all leading to derived safety requirements. For this thesis, these underlying steps are assumed to have contributed to the mishap risk table below and subsequent safety requirements. The mishap risk is determined according to the resultant mishap and then carried over to the associated hazard(s). Mishap risk will be determined in accordance with the Joint Software System Safety Committee Software System Safety Handbook [2], utilizing the hazard severity and hazard probability to determine a Hazard Risk Index (HRI) rating whilst using engineering judgment to combine risk elements of software using the Software Hazard Criticality Matrix (SHCM).

ID	Date	Severity	Probability	HRI	Comments
H1	15Jul04	Catastrophic	Frequent	1	Unacceptable risk. Safety requirements to lower risk.
	30Sep04	Catastrophic	Probable	2	Risk lowered after review by SME, software control level lower than anticipated.
H2	15Jul04	Catastrophic	Remote	8	Marginal risk. Based on current rocket motor success.
	10Oct04	Catastrophic	Improbable	10	Minimum risk. Risk lowered after review by SME.

ID	Date	Severity	Probability	HRI	Comments
H3	15Jul04	Catastrophic	Occasional	4	Unacceptable risk. Current SAM systems not as software-intensive as RASAM. Safety requirements to lower risk.
	08Oct04	Catastrophic	Probable	2	Risk raised after comparison to current non-software-intensive systems. Safety requirements not yet sufficient.
H4	15Jul04	Critical	Occasional	6	Marginal risk. Based on current launcher technology. RASAM involves more software control.
H5	15Jul04	Critical	Remote	12	Minimum risk. Based on current electrical safety technologies.
H6	15Jul04	Critical	Improbable	15	Minimum risk. Assumes trained maintenance personnel.
H7	15Jul04	Catastrophic	Remote	8	Marginal risk. Based on current toxic material containment.

ID	Date	Severity	Probability	HRI	Comments
H8	15Jul04	Critical	Remote	12	Minimum risk. Remote possibility that personnel will be near debris. Equipment and environment not affected.
H9	15Jul04	Catastrophic	Probable	2	Unacceptable risk. Safety requirements to reduce risk.
	09Aug04	Catastrophic	Occasional	4	Risk lowered after preliminary safety requirements reviewed.
H10	15Jul04	Catastrophic	Occasional	4	Unacceptable risk. Safety requirements to reduce risk.
	25Jul04	Catastrophic	Probable	2	Risk raised due to review by SME.
H11	15Jul04	Catastrophic	Probable	2	Unacceptable risk. Safety requirements to reduce risk.
H12	15Jul04	Catastrophic	Probable	2	Unacceptable risk. Safety requirements to reduce risk.
H13	15Jul04	Marginal	Occasional	13	Minimum risk. Assumes trained maintenance personnel.

ID	Date	Severity	Probability	HRI	Comments
H14	15Jul04	Catastrophic	Probable	2	Unacceptable risk. Based on current SAM systems.

Table 13. RASAM Preliminary Hazard Analysis

H. RASAM SOFTWARE SAFETY REQUIREMENTS

The results of the PHA typically identify the preliminary system components that can contribute to mishaps/hazards. It has been assumed that, without explicitly undertaking an analysis method (Failure Modes and Effects Analysis, Fault Tree Analysis, Event Tree Analysis, etc.) in this thesis, the results of such a method are available. Therefore, the safety-critical system components have been identified and the following safety requirements are developed to reduce the probability of causal factors contributing to the mishap. Again, this will not be a “complete” exercise, it will only illustrate a portion of safety requirements. As this documentation is being formed in the conceptual design phase, sufficient design knowledge is not present to sufficiently identify software safety requirements. This will be conducted during the following design phases.

The following requirements are derived safety requirements (Safety Derived – SD*n*) identified to directly address hazard H9 and partially H14:

SD.1. WCS shall maintain positive control of launch-related commands.

SD.2. The design of the interfaces shall reduce the probability of erroneous target designation and launch-related messages due to one-, two-, or three-bit errors to $<1x10^{-9}$.

SD.3. The LSC design shall verify the correct sequence of launch related commands.

SD.4. The WCS shall validate target designation commands from C&C.

SD.5. The LSC shall verify safety-related interlocks prior to complying with launch related orders.

In order to address the probability of hazard H1 occurring unintentionally, the following generic safety requirement (Safety Generic – SG*n*) is identified:

SG.1. Warhead fuze design shall be in accordance with MIL-STD-1316: Safety Criteria for Fuze Design.

Addressing the probability of hazard H2, the following generic safety requirement is identified:

SG.2. Rocket propulsion system shall be designed in accordance with MIL-STD-1901A: Safety Criteria for Munition Rocket and Missile Motor Ignition System Design.

I. RASAM SSRTM

Although the above requirements are not yet specific “software” requirements, they will form the basis of the SSRTM. In fact, these requirements will form the System Safety Requirements Traceability Matrix from which the Software Safety Requirements Traceability Matrix will be defined. The following matrix will be in the same form as the SSRTM, it will simply not yet address specific software components.

Safety Rq. ID	Hazard ID	Op. Rq. ID
SD1	H9	OR4, OR6, OR8
SD2	H9	OR3, OR4, OR7, OR8
SD3	H9	OR6, OR8
SD4	H9, H14	OR3, OR4, OR7, OR8
SD5	H9	OR7, OR8
SG1	H9	OR*n*
SG2	H9	OR*n*

Table 14. RASAM SSRTM

The above SSRTM is only a rudimentary example. The key element is that traceability from any given safety requirement can be made back to identified hazards. At this stage of development, it can usually be assumed that the identified hazards are valid themselves. The processes (misuse cases, previous experience) used to identify the hazards, along with application of this framework, will contribute to validity of the hazards. Traceability in the safety sense primarily considers traceability to an identified hazard which is then traceable (through the analysis and identification methods) to system functionality.

LIST OF REFERENCES

[1] MIL-STD-882D, Standard Practice for System Safety, DoD, 2000.

[2] Joint Software System Safety Committee, Software System Safety Handbook: A Technical and Managerial Team Approach, Joint Services Computer Resource Management Group, U.S. Navy, U.S. Army, U.S. Air Force, 1999.

[3] Institut de l'Audiovisuel et des Télécommunications en Europe, Software intensive systems in the future, version 5, September 2005, pg 5.

[4] IEEE Std. 1012-2004, IEEE Standard for Software Verification and Validation, IEEE, 2004.

[5] B. S. Blanchard, and W. J. Fabrycky, Systems Engineering and Analysis, 4th Edition, Pearson Prentice Hall, 2006.

[6] D. Drusinsky, J.B. Michael and M. Shing, A Framework for Computer-Aided Validation, Innovations in Systems and Software Engineering, 4(2), pp. 161 - 168, June 2008.

[7] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE 1990.

[8] J. C. Munson, Software Engineering Measurement, Auerbach Publications, 2003.

[9] M. W. Whalen, M. P. E. Heimdahl, A. Rajan, S. P. Miller, "Coverage Metrics for Requirements-Based Testing," *ISSTA*, pp. 25 – 35, 2006.

[10] S. Tasiran and K. Keutzer, "Coverage Metrics for Functional Validation of Hardware Designs," *IEEE Design and Test of Computers*, July – August, 2001.

[11] P. J. Pingree, E. Mikk, G. J. Holzmann, M. H. Smith, and D. Dams, "Validation of Mission Critical Software Design and Implementation Using Model Checking," *IEEE*, 2002.

[12] J. Wing, "A Specifier's Introduction to Formal Methods," *IEEE Computer,* vol. 23, no. 9, pp. 8-24, September 1990.

[13] B. Boehm and P. Bose, A Collaborative Spiral Software Process Model Based on Theory W, *Proceedings of the Third International Conference on the Software Process*, 1994, pp 59 – 68.

[14] N. Leveson, Safeware: System Safety and Computers, Addison-Wesley, 1995, pp 29, 249 – 260, 359.

[15] R. Weaver, “The Safety of Software – Constructing and Assuring Arguments,” Ph.D. dissertation, University of York, Department of Computer Science, September 2003, pp 81- 85.

[16] K. Gödel, Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik 38*, pp. 173 – 198, 1931.

[17] V. R. Basili, G. Caldiera and H. Dieter Rombach, “The Goal Question Metric Approach,” in *Encyclopedia of Software Engineering*, Vol. 1, John Wiley & Sons, 1994, pp. 528-532.

[18] V. R. Basili and D. M. Weiss, “A Methodology for Collecting Valid Software Engineering Data,” *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, pp. 728-738, November 1984.

[19] P. Berander and P. J□nsson, A Goal Question Metric Based Approach for Efficient Measurement Framework Definition. *Proceedings of the International Symposium on Empirical Software Engineering*, September 2006.

[20] V. R. Basili and H. D. Rombach, “The TAME Project: Towards Improvement-Oriented Software Environments,” *IEEE Transactions on Software Engineering*, vol. SE-14, no. 6, pp. 758 – 753, June 1988.

[21] W. Royce, *Managing the Development of Large Software Systems*, *Proceedings of IEEE WESCON*, pp. 1–9, August 1970.

[22] B. Boehm, A Spiral Model of Software Development and Enhancement, *IEEE Computer*, May 1988, pp. 61 – 72.

[23] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, Prentice Hall PTR, 2005.

[24] R. Weaver, J. Fenn, and T. Kelly, “A Pragmatic Approach to Reasoning about the Assurance of Safety Arguments,” in proceedings of the *8th Australian Workshop on Safety Critical Systems and Software (SCS’03)*, Canberra, 2003, vol. 33, pp. 57 – 67.

[25] V. Basili, K. Dangle, L. Esker, and F. Marotta, “Gaining Early Insight into Software Safety: Measures of Potential Problems and Risks,” presented at Systems and Software Technology Conference, June 2007.

[26] P. Redmond, “A System of Systems Interface Hazard Analysis Technique,” M.S. thesis, Naval Postgraduate School, Monterey, CA, March 2007.

[27] B. Warren, “A Framework For Software Reuse In Safety-Critical System of Systems,” M.S. thesis, Naval Postgraduate School, Monterey, CA, March 2008.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Library
   Australian Defence Force Academy
   Cambell, Australian Capital Territory, Australia

4. Squadron Leader Derek Reinhardt
   Royal Australian Air Force
   RAAF Williams, Laverton, Australia

5. Mr Arch McKinlay
   Naval Ordnance Safety and Security Activity
   Indiana Head, Maryland

6. Professor Bret Michael
   Naval Postgraduate School
   Monterey, California

7. Professor Man-Tak Shing
   Naval Postgraduate School
   Monterey, California

8. Dr. Tim Kelly
   University of York
   Heslington, York, United Kingdom

9. Dr. Rob Weaver
   University of York
   Heslington, York, United Kingdom

10. Mr. Michael Brown
    EG&G Technical Services, Inc.
    Fresno, California

11. Mr. Marcus Fisher
    NASA IV&V Facility
    Fairmont, West Virginia

12. Mr. Wee Kok Ling
Defense Science & Technology Agency
Singapore

13. Mr. Mark Wessman
Wessman Consultancy Group, Inc.
Springfield, Virginia

14. Mr. Lim Horng Leong
Defense Science & Technology Agency
Singapore

15. Mr. Nickolas Guertin
NAVSEA, PEO IWS 7
Washington Navy Yard, DC

16. Mr. John Harauz
Jonic Systems Engineering, Inc.
Willlowdale, Ontario, Canada

17. Dr. Butch Caffall
NASA IV&V Facility
Fairmont, West Virginia